Global Information Assurance Certification Paper

Remotely Accessing Sensitive Resources

GIAC GSEC Gold Certification

Author: Jason Ragland, jason@infosec.utexas.edu
Advisor: Rick Wanner

Accepted: January 18th, 2010

Abstract

It is a fact of life that most companies' workforces are becoming increasingly mobile. Because of this, vast amounts of sensitive data are exposed to unauthorized access and misuse through loss, theft, or confiscation of portable devices. This represents a very serious problem for companies. A common approach to dealing with this threat has been to use encryption to protect the data, but this alone does not guarantee that the data will not be compromised or negate the consequences of breaches of confidentiality. Additionally, data encrypted at rest must be decrypted so that it can be worked with, which exposes it to malware and network-based attacks. Using well-documented technologies that individual users can implement, such as SSH, remote desktop applications, and USB storage devices, sensitive data can instead be kept on network accessible workstations or servers. The implementation of these technologies is adaptable to suit a wide range of environments and needs while protecting data from network and local threats. With these strategies, local copies of data are not necessary so loss of a device exposes nothing.

1. Introduction

Often travelers require access to digital resources to perform work from off-site locations such as conferences, hotels, and homes. These resources can include emails, research, medical, financial data, server management applications, or any number of other things that may have a very high need for confidentiality and integrity. The acceptable methods for access vary based on a variety of factors such as size, complexity, available types of network connectivity, and bandwidth. Access to email is often easily provided via a secure website and a password, for example. If the resource consists of gigabytes of research data, it isn't as simple.

A popular solution to this problem is for users to travel with a laptop containing copies of the sensitive data they need to work with. While this does mitigate the risks of having to transfer large quantities of data over the network and possibly even the need for network connectivity at all, it exposes the data to a whole new set of threats. Intangible threats like vulnerabilities in installed software packages, malware or file-sharing applications, and brute force attacks can and have exposed countless amounts of data unintentionally. Even devices that are well secured from a software standpoint are subject to higher likelihood of loss or theft thanks to their size and portability. In fact, Safeware Insurance announced that, of all computer electronics claims, theft was the second most likely cause of loss for their policyholders in 2007, and that the number of reported thefts had increased 29% over the course of the year (PRWeb, 2008). Dealing with incidents involving unauthorized disclosure of data is costly, both in terms of notification and reputation (Brelsford, n.d.)(Mark, 2005). Many states have or are considering adopting mandatory disclosure laws that may increase these costs. Additionally, various regulatory bodies and industry consortiums can levy fines or even criminal charges depending on the situation. Merely encrypting the data alone does not remove all of these consequences (Dougherty, 2008)(BBC, 2008)(Public Service, 2009).

Even if encrypted, once the data falls into unauthorized hands, it becomes impossible to track the existence of any copies or ensure that the data hasn't been compromised. Advances in technology and undiscovered vulnerabilities in encryption

implementations may allow copies of this data to be decrypted and used for malicious purposes at some point in the future. Consider the example of the DES encryption algorithm that was widely used and a Federal Information Processing Standard (FIPS) for twenty-seven years. In 1997, twenty-two years after it became a FIPS standard, it was publicly broken by a distributed computing project that successfully brute forced the encryption key (Curtin & Dolske, 1998). The time required to break DES has decreased dramatically over the years. Anything encrypted with DES can now be decrypted in a matter of days with a relatively small investment in hardware. For some types of data with a short usable lifespan, the risk of this occurring again with another type of encryption may not pose much of a threat, but many types of data remain sensitive for considerable periods of time. Even worse, it would be very hard to detect that this breach of sensitive data has happened until after the data is misused.

International travel brings added risk as customs agents in both the United States and other countries become more technologically savvy. U.S. Customs and Border Protection (CBP) have increasingly copied the contents of encrypted drives, compelled users to divulge credentials, and even outright confiscated computer equipment (Nakashima, 2008). While CBP claims that all data copied from these laptops is stored securely and kept confidential, there is no available evidence to support this. Regardless of how the data is secured, it is still in the hands of unauthorized and potentially untrustworthy people. As a direct result, many travel advocacy groups now advise international travelers not to take laptops with them, or to only carry laptops without any sensitive data abroad (Schoultz, 2007)(Jonas, 2008)(Baker, 2008). It's not hard to imagine other countries following the lead of the US at their own borders.

If users opt to travel without laptops, they're forced to use whatever happens to be available at a given location. The security of public workstations and kiosks can be a huge unknown. They may be configured insecurely, be unpatched, have spyware and file-sharing applications installed, or be infected with keyloggers and other malware. For example, in 2003, a customer placed a keylogger on public computers in thirteen Kinko's stores in the New York City area and managed to obtain over 450 passwords to online bank accounts. His actions were only discovered after he used a set of stolen credentials to remotely control a computer while the victim was actively using it (Poulsen, 2003).

Author Name, email@addressinfosec.utexas.edu

Furthermore, employees may have also intentionally compromised the computers for their own nefarious purposes. Seldom do users have the required access rights to verify the machine is secure or to isolate their data from any existing malware.

Whether users travel with a laptop or use a computer on-site, the network they connect to at the remote site also poses a threat. Malicious users may attempt to compromise other hosts sharing the network or capture traffic for later analysis. Malware on infected computers, seeking to propagate, will often scan for and attack other hosts. Besides other users, the employees administering the network may be sniffing traffic or have set the network up insecurely to take advantage of their users. Even using an encrypted communications protocol is not a guarantee of safety, as illustrated by the recent flaw discovered in the pseudo-random number generator used by OpenSSL in Debian. This flaw created certain encryption keys that were far weaker than expected and susceptible to simple brute-force attack (US-CERT/NIST, 2008).

Still, accessing data remotely instead of carrying copies on portable devices or removable media has some very strong advantages. The risks posed by foreign networks and the unknown state of kiosks and public workstations can be countered to some extent. The fact that not storing sensitive data on a portable device makes theft or loss of that device a mere inconvenience instead of a potentially disastrous event, resulting in bad press or worse, makes this an option worth considering. Additionally, law enforcement and border protection agents can carry out their jobs without users being obligated to place sensitive data at risk. There will be no worries that someone with a copy of an encrypted set of data will be able to decrypt and misuse it. However, allowing remote access to data requires preparation in advance with knowledge of what specific data will need to be accessed and details about the remote environment.

2. Threats

Before discussing some of the possibilities for accessing data remotely, it is important to define the threats involved.

2.1. Malware

All types of malware either deceive users into installing them or use exploits to install themselves without user consent or awareness. Viruses spread when users unknowingly execute an infected file while trying to run an otherwise innocuous application. Trojan horses are subtlety different from viruses in that, although they still require users to execute the application and infect their own computers, they masquerade as other applications entirely. Once infected with either, the attacker is able to do anything that the users themselves are capable of doing with the permission level of their user account. This includes installation of additional malware to provide enhanced functionality not present in the original malware. Worms, unlike viruses and trojan horses, are capable of replicating without any user assistance by taking advantage of exploits in software and operating systems. By using exploits to infect machines, worms may actually have a higher level of access to the system and as such can perform actions that users cannot. This is because the permissions granted to a worm are based upon the exploit used rather than tied to a user account, as is the case with viruses. Malicious applications are threats to users looking for a safe local host with which to start a remote session as the host cannot be counted on to behave predictably or reliably. The integrity of any data processed or stored on an infected local host during a remote session is questionable.

Spyware is another type of malware that exists primarily to collect information about users. The information collected may be used by the attackers to compromise other computers or used against users directly (e.g. identity theft). This type of malware often changes settings and interferes with users in such a way that prevents them from removing the malware. The key difference between spyware and the other malware, however, is that spyware is less likely to obviously alter or destroy any data (i.e. the integrity of the data is unchanged) and more likely to capture the data and any actions taken by the user.

2.2. Keyloggers

The threat of keyloggers is present on just about any computer. They can be installed in person by malicious users with legitimate access or remotely if the computer

is not well secured and susceptible to any other form of malware. Trojans and worms have been known to install keyloggers so that the attacker can obtain more information with which to perpetrate other crimes, such as identify theft (Claburn, 2005). Very simple keyloggers may just record all keystrokes entered by a user. More complex ones can target specific web sites and applications, access a user's clipboard, and even take screenshots to defeat on-screen keyboards and other entry mechanisms.

Detection of keyloggers can be quite difficult as these applications are designed to be stealthy and may not show up as active applications or in process monitors. Due to their unobtrusiveness, they tend not to interfere with normal operation of the computer so typically users only discover they have fallen victim after the attacker has used information recorded by the application in some way (Grebennikov, 2007). Keyloggers can even take the form of hardware devices, which would circumvent any software intended to detect or prevent installation of such malware. For users connecting to remote computers to access sensitive data, keyloggers may provide everything required for the attacker to log into the remote computer and access the same data later.

2.3. File Sharing Applications

The threat posed by file sharing, or peer-to-peer, applications manifests itself in a number of ways. These programs are often configured by default to share the contents of common document and media folders without requiring any interaction by the user. This frequently catches people by surprise when they discover that their sensitive documents are available for download by other users (ZapShares Inc., 2009). Users may also unknowingly place files they are working with in shared folders. Even the person who installed the application is apt to do this, as there are no consistent means across these programs to identify shared folders outside of the applications themselves. Additionally, after downloading something, it remains available to other users of the application until the user takes specific action to make it not shared, something which again may not be clearly communicated (P2P Security, 2008).

Use of these applications also increases the risk of compromise by malware, as there are no guarantees that the files being downloaded are what they claim to be. It is trivial for a malicious user to share a trojan or something infected with a virus in the

hopes that other people will download it and compromise their own computers. Even the mere act of installing a file-sharing application is a serious risk as many of these applications also install spyware and adware (Aftab, 2004).

2.4. Unencrypted Sessions

Unencrypted sessions refer to network sessions that are not protected by any form of encryption. Anyone who intercepts the packet data sent by the transport protocol can reassemble the segments and obtain the data. Essentially they can read everything sent from one device to another during the session. This can be accomplished by having access to a device in-line between the legitimate participants of the session, such as a router or switch. The attacker can capture the data as it is sent to this device on its way to the destination device. Another possibility is to use malware or an exploit like ARP spoofing or DNS cache poisoning to configure a computer to route packets to another device controlled by the attacker. In such a scenario, when the victim's computer attempts to contact the legitimate device, the traffic is instead routed to the attacker's device where the packets can be captured and manipulated. ARP spoofing is limited to the hosts on a local network and pretty much can only be used to sniff packets or conduct man-in-the-middle and denial-of-service attacks. DNS cache poisoning, by comparison, is not only capable of those same types of attacks, but can also be used to spread malware and phish for user data, such as credentials. It can be very difficult for victims to determine when these types of attacks are occurring.

Wireless networks present an increased risk of packet sniffing as the data is broadcast and anyone within range can capture it. Many freely provided, publicly accessible wireless networks do not utilize any encryption options in order to keep the configuration simple for users (AirTight Networks, n.d.)(NCSA, n.d.). Even the most likely encryption that you might find on such a network, WEP and WPA-PSK, only provide security against people who are not authenticated to the wireless network. Network traffic appears unencrypted between the client and the access point for all users on the wireless network. This means that any user can intercept and examine the contents of any packet sent from another client. People who are not authorized to connect to the network can still collect the encrypted packets, and WEP has vulnerabilities that make it

possible to break the encryption given a large enough sampling of data (Ossmann, 2004). WPA-PSK is more secure and does not share that specific issue, but is instead vulnerable to brute force attacks on the pre-shared key (Fogie, 2005). This could result in unauthorized individuals being able to connect to the network, where they may attempt to capture traffic or conduct attacks against legitimate clients. These issues, combined with the ease of setting up rogue wireless networks, make public hotspots very good places for malicious users to attack unsuspecting people (AirDefense, 2007)(AirDefense, 2008).

Unencrypted sessions can be dangerous because they not only lead to unintended information disclosure via packet sniffing, but they also make more serious attacks, like session hijacking, easier to perform.

2.5. Session Hijacking

Session hijacking is a type of attack where a malicious user takes over an existing session from a legitimate user. This is done by compromising the mechanisms used by applications for session management. With web applications for example, session management is often done through the use of session cookies, which contain the user's session ID. It is very common to see web applications protect login pages with SSL to safeguard user account credentials from sniffing, but not protect other parts of the site after the user has authenticated (Mills, 2008). If encryption is not utilized for the entire network session, and the cookie is not properly marked with the secure flag, when the session cookie is sent to back to the server as a part of any future requests from the user's client anyone capable of capturing the packet traffic can obtain it. Cross-site scripting vulnerabilities in web applications, where attackers take advantage of unverified inputs in web pages to inject their own code, provide another way for an attacker to capture a session cookie. This exploit, which does not even require the ability to intercept network traffic, allows the attacker to instruct the user's browser to return a copy of the session cookie (OWASP, 2009).

Applications that rely upon the session management controls built into TCP are also subject to hijacking attacks. When a TCP session is created, one of the parameters defined by the party initiating the connection is the sequence number. The sequence number starts as a random number in the header of the first packet sent, but increments in

a predictable manner with each successive packet. Even though packets are sent in sequential order over a session, they may arrive at the destination out of order due to network issues, so the sequence number provides a way to determine the correct order of the packets. When the destination receives a packet, it will acknowledge receipt of that packet and indicate the sequence number that it is expecting to see in the next packet. This back and forth exchange will occur over the duration of the session with the sequence number incrementing every time a new packet is received. If the attacker can capture this traffic, determine the next sequence number, create a packet with this sequence number, and send it to the server before the client, then the server will accept the attacker's packet and drop any other packets using that sequence number as duplicates. The client's session becomes desynchronized in that it keeps sending the same packet to the server over and over waiting for an acknowledgement, but the packet doesn't have the sequence number the server is expecting so the server drops it. The attacker can then conduct an ARP spoofing attack to route packets from the server to another device in order to both capture responses from the server and prevent the server from sending any further packets to the client. At this point the attacker has successfully hijacked the session, and knowing the next expected sequence number, is able to continue sending new packets which will be accepted by the server as valid.

Man-in-the middle attacks are another form of session hijacking wherein the attacker makes an independent connection with each participant of a network session, intercepts all of the communications between the victims, and injects new messages into the session that are delivered to each party. The result is that the participants believe they are communicating directly with each other but in reality the attacker is in between them and controls the entire conversation. In order to accomplish this, the client needs to be tricked into communicating with a device controlled by the attacker, which appears to the client as the intended recipient. Like previous examples, this could be arranged with DNS cache poisoning or ARP spoofing. Then the attacker makes a connection to the actual recipient and passes responses between both parties. Successfully done, this not only allows the attacker to capture any data that the victims are attempting to send to each other, but also allows the attacker to direct the conversation to obtain data that he is interested in. This can even be used to defeat attempts between the two parties to

negotiate an encrypted session by intercepting and substituting the public keys sent by each with public keys from a key pair controlled by the attacker (RSA Laboratories, n.d.).

For users considering working with data remotely, there are two things to understand about man-in-the-middle attacks. First, public wireless networks present a great opportunity for those interested in conducting man-in-the-middle attacks as it is relatively easy for an attacker to set up a rogue access point that looks legitimate and wait for users to join. This sets up part of the attack, wherein the user connects to a device controlled by the attacker and the attacker is able to intercept the user's traffic. Second, any application that does not do an adequate job of verifying the identity of a remote host when setting up a session is vulnerable to this type of attack.

2.6. Data Left Behind

User activity often results in temporary data being created or stored on a computer. For instance, the common task of browsing the web may result in the creation of and modification to a number of files that may remain on the machine for some length of time. Cached files and cookies are two examples of files that are commonly created from browsing web sites. Cookies in particular may contain sensitive session information or other personal data related to a user's visit to a site. Improperly configured sites may not instruct the browser to avoid caching specific pages, which can lead to sensitive data being stored in the cache folder. Additionally, many browsers keep track of the sites visited by users with history and favorites databases. Any future users of the computer may have access to this information.

Many other tasks similarly leave varying quantities of data behind. Email clients may log connection specific details as well as store credentials for accessing email accounts and even the messages themselves. Document editing software often creates temporary copies of opened documents. Many applications, after experiencing an error, will create event log entries that could contain user data. Even system event logs will usually detail at a minimum when users log in and out.

How much data is left behind as a result of using any computer is hard to determine as it depends on the configuration of the computer and applications and what exact tasks are performed. It is probably fair to say that, without administrative

credentials, there is simply no way for users to be confident that no traces of their activities have been left behind. Having administrative credentials allows users to look for and clear out evidence of their activities in far more places, but there are still no guarantees.

3. Defenses

There is no one solution to all of the risks faced by users attempting to work with data from remote locations. What is possible depends upon the environment at the remote site. The more that is known about this environment in advance, the better a solution can be tailored to address specific needs or shortcomings.

In this section some of the many options are discussed. Under each topic, there is a short explanation of what the option applies to in order to help find useful and relevant information more quickly. The solutions are targeted to what an individual user can do without reliance upon or assistance from an IT department, although some may be adaptable for use on a larger scale.

3.1. Virtual Private Networks

Defends against: Session hijacking, Unencrypted sessions

As explained earlier, unencrypted sessions can lead to privacy and information disclosure issues, as well as play a large role in enabling session hijacking attacks. To defend against this, the communication channel between the local and remote computers should be encrypted somehow. Many web applications requiring confidentiality have implemented SSL to meet this need, which is, fortunately, easy to do. For native applications, on the other hand, safeguarding the traffic between client and servers largely depends on what, if any, measures the developers decide to implement and the inherent security of whatever communications protocol is used. Applications often end up with inadequate security precautions though, because the developers lack both knowledge of the vulnerabilities present and an understanding of how to defend against the threats faced (Christey & Martin, 2007)(Farber, 2007). Additionally, many of the protocols in use today predate current knowledge about common threats and countermeasures.

Design decisions that were considered acceptable years ago, such as omitting options for encryption, now present serious risks (Chambers, Dolske, & Iyer, n.d.).

There are some technologies users can take advantage of to encrypt communications between applications and servers regardless of how they were designed. One option is to set up a Virtual Private Network (VPN). VPNs are virtual networks implemented in software on top of existing networks. The purpose is to extend a network over one or more other networks. A computer connected to a VPN is assigned an IP address on a virtual interface over which it is able to communicate with computers on the remote network as if it were connected locally. With a standard VPN, all traffic from the client is routed over the private network, even if it is not intended for other hosts on that network. This can cause performance slowdowns and waste bandwidth if the traffic wasn't originally intended for the private network. Split VPNs were developed to counter this issue. In a split VPN, only the traffic destined for the private network is secured and sent over the VPN interface. All other traffic is routed to its destination from the client machine as if the VPN did not exist.

A caveat to using VPNs is that they do not provide protection from endpoint to endpoint, but from the VPN client to the VPN server, where both the client and server may be hardware devices or software applications separate from the host endpoint and the destination endpoint. The traffic is sent decrypted to the client VPN device or application, encrypted while in transit to the remote network, decrypted at the VPN server, and then sent decrypted to its final destination if the VPN server is not also the destination server. This may not be an issue if the destination host is on the same private network as the VPN server or if the encapsulated protocol is also encrypted.

Despite their strengths, VPNs are generally not an attractive solution for individual users. Using a VPN often requires that client software or drivers be installed to negotiate and manage the encrypted connection. Installing software and drivers requires administrative credentials in modern operating systems, which is something users are very unlikely to have on a remote workstation such as a public computer or kiosk. In such a situation, the user would be unable to configure the client side of the VPN and therefore would be unable to connect securely to the remote server to access the

desired data. VPNs become a better option if users travel with laptops instead as these can be configured in advance. Still, this is something that most users will be unable to do without assistance.

3.2. Secure Shell Tunneling

Defends against: Session hijacking, Unencrypted sessions

Secure Shell (SSH) is mostly known as a tool to connect to and run commands on remote computers. However, SSH can also be used to encrypt traffic for almost any application, protecting against the same types of network threats as VPNs. Compared to VPNs, using SSH in this manner is far easier to setup, requires no administrative credentials on the client machine, and the software client is included by default or easily obtainable on most operating systems.

The idea is to use SSH on the client to create a tunnel to a remote SSH server. This remote machine may be the desired destination for the client or just an intermediary server. Instead of the client connecting directly to the destination server and sending data potentially unencrypted over untrusted networks, the connection is made to the SSH tunnel. Then the client sends traffic to the tunnel, where it is encrypted by SSH and forwarded to the SSH server. Once there, it is decrypted and, if the SSH server is not the intended destination, a connection is made to the remote machine on behalf of the client. So, like a VPN, this provides a means to get the data from one machine to another safely, but it's point-to-point and does not extend the private network at all. Note that this shares a drawback previously discussed with VPNs. The traffic is encrypted only to the SSH server and then proceeds from that computer, unprotected by SSH, over the remote network to its destination. It can be much easier, however, to have an SSH service running on the endpoint server itself to remediate this whereas with VPNs remediation depends on other factors such as whether the VPN is hardware or software based. The closer the SSH server can be to the destination server on the network, the better.

There are a couple of options for using SSH tunnels depending upon what traffic is to be encrypted. The first is to create a dynamic application level port-forwarding tunnel. In this scenario SSH acts as a SOCKS proxy. Individual applications or the operating system as a whole can be configured to direct network connections to the SSH

proxy server on the client. The connections are forwarded over the secure tunnel and the remote server makes the actual connection to the specified resource. Before a user will be able to do this, the server must be configured to allow tunneling. The location of SSH server configuration file will vary by operating system, but the parameter that must be set to "yes" is PermitTunnel. Once this is done and the SSH server process has been restarted, the command to establish a tunnel in proxy server mode looks like this:

```
ssh user@sshserver -D localport –N
```

The parameter "–D" instructs SSH to allocate a socket on the client to listen on the port specified in the "localport" parameter. Adding "-N" simply tells SSH not to run any remote commands, but only forward the specified ports. The applications or the operating system must be configured to direct traffic to this local port for it to be encrypted by SSH. The process for this will be different for each application, but many have a network specific configuration interface that allows a user to specify a SOCKS proxy that is to be used by the application. Otherwise, the operating system may be configured to use the proxy for all traffic, which will work, but is less desirable since even data that does not need to be protected will be forwarded through the tunnel and encrypted. Using SSH in this manner is the best option when traffic from multiple applications needs to be protected, when the traffic is sent over many connections on different ports, or when the ports used are ephemeral and subject to change often. A good example where this type of SSH tunnel would be suitable is web browsing. It would be arduous to secure a typical web browsing session with other methods since connections to many different servers over multiple ports are made. Additionally, web browsers have settings for configuring network proxies separate from the general operating system network configuration options, which may not be accessible by users.

The second option for SSH tunneling is to simply forward a local port on the client machine to a remote port on the server. The command to do that would look like:

```
ssh user@sshserver -L localport:remoteserver:remoteport -N
```

In this configuration, SSH will not act as a proxy. When the tunnel is established, the SSH server opens a connection and forwards all traffic sent from the client over the tunnel to the remote server and remote port specified. So, unlike the previous

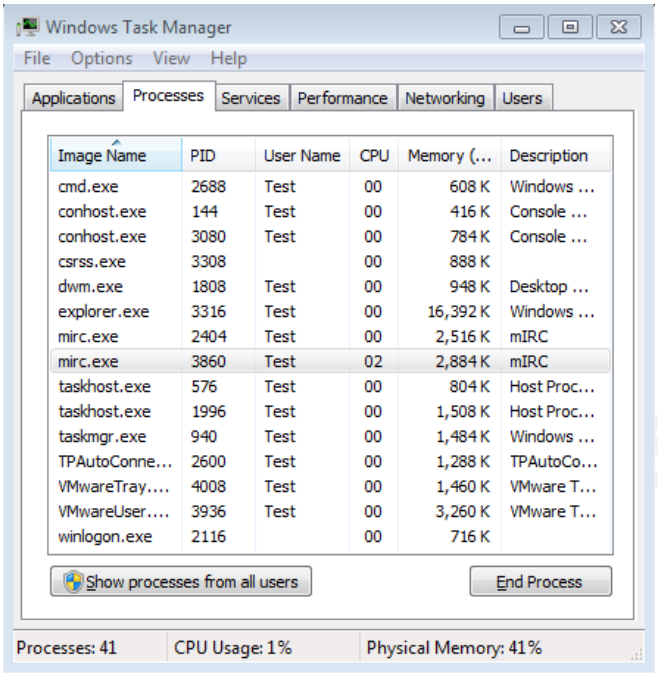
configuration, SSH is not determining what the destination of the traffic sent over the tunnel is as this has already been provided. This means that forwarding multiple ports or forwarding traffic to different destinations requires more tunnels to be setup – one for each port and destination. Consider a situation where a user needs to send and receive email. Depending on what connection protocols are accepted by the email server, securing this traffic would require two SSH tunnels, one for POP3 on port 110 or IMAP on 143, and one for SMTP on port 25. However, the application configuration would be identical except instead of specifying the real email server, localhost would be used. This is an easy option to setup with most applications, including ones that do not have an option to specify a proxy server.

Determining which ports to forward with SSH can be tricky. Almost all operating systems have a version of the netstat application that will provide a list of all open network ports and their current state. The difficulty is in mapping the open ports to the application using that port. For Linux, the netstat command with the parameters "-nap" will output a list of open network connections and the process ID (PID) and program name bound to each port. The output should look something like the excerpt below.

```
jason@Durandel:~$ netstat -nap
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp    0    0 0.0.0.0:2022            0.0.0.0:*               LISTEN      -
tcp    0    0 127.0.0.1:631           0.0.0.0:*               LISTEN      -
tcp    0    0 10.1.1.101:60175        65.55.11.163:80         CLOSE_WAIT  -
tcp    38   0 10.1.1.101:55653        174.36.30.67:443        CLOSE_WAIT  4136/dropbox
tcp    0    0 10.1.1.101:41149        172.16.78.107:1462      ESTABLISHED -
tcp    38   0 10.1.1.101:33333        96.17.243.51:443        CLOSE_WAIT  21910/vmware
tcp    0    0 10.1.1.101:46135        207.46.193.254:80       CLOSE_WAIT  -
tcp    38   0 10.1.1.101:33331        96.17.243.51:443        CLOSE_WAIT  21910/vmware
tcp    0    0 10.1.1.101:45209        172.16.80.58:49155      ESTABLISHED -
tcp    0    0 10.1.1.101:37240        209.107.211.121:80      CLOSE_WAIT  -
tcp    38   0 10.1.1.101:55501        174.36.30.68:443        CLOSE_WAIT  4136/dropbox
tcp    0    0 10.1.1.101:35060        10.1.1.100:24800        ESTABLISHED 4092/synergyc
tcp    38   0 10.1.1.101:33332        96.17.243.51:443        CLOSE_WAIT  21910/vmware
```

The foreign address field identifies the port on the remote server listening for incoming connection requests for the application. With this output, users will know what values to specify for both the "remoteserver" and "remoteport" parameters when creating the tunnel. Depending on whether the application is being run as a service, root credentials may be required to show the PID. Mac users can run the lsof command, which provides information about files opened by processes, with the parameters "–i" and "+M" to

output a similar list. These parameters instruct lsof to select the listing of all Internet files and display any portmapper registrations that may exist. For Windows, the process works somewhat differently. First the PID of the application in question needs to be identified and then this can be used with netstat to find the ports opened by that process. Process IDs can be identified through the Task Manager interface or through the command line application, tasklist.



Once the PID is known, pipe the output of netstat with the "-ao" parameters to findstr to quickly narrow the results.

```
netstat -ao | findstr [PID]
```

What this command will do is direct the output of netstat, which is configured to list all active connections and ports as well as the PID bound to each, to the findstr command, which is set to filter on strings containing the desired PID.

```
C:\Users\Test>netstat -ao | findstr 3860

  TCP    192.168.20.130:49800   ec2-79-125-11-206:6667  ESTABLISHED     3860
```

In some cases administrative or root access will be required to do this, so it is important that users have this information before attempting to setup a tunnel from a remote site as they are not likely to have this level of access on public kiosks or workstations.

Author Name, email@addressinfosec.utexas.edu

Upon connecting to an SSH server for the first time from a machine, users will be prompted to verify the identity of the server. This provides an important protection against man-in-the-middle attacks, as, if the host key displayed matches the host key of the actual server, then users can be confident that a secure, encrypted session has been successfully established with the desired host (Hatch, 2004). The host key of an SSH server can be obtained with the ssh-keygen command as demonstrated below.

```
jason@Durandel:~$ ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key.pub
2048 46:a7:9b:da:9f:54:31:39:e1:b7:03:59:52:d6:40:7a
      /etc/ssh/ssh_host_rsa_key.pub (RSA)
```

The "-f" parameter specifies the key file while the "-l" instructs ssh-keygen to display the fingerprint of indicated key file. If users have this fingerprint prior to connecting, verifying the host is as simple as checking the value displayed with what the user has. In cases where a user does not know the fingerprint in advance, the identity of the host can still be verified by ensuring the fingerprint displayed when connecting and the fingerprint of the key file on the host, determined after connecting using the same command above, match.

SSH tunnels allow users to do one additional thing that can be very handy for connecting to servers remotely: firewall evasion. In some cases, users at remote sites may find that network connectivity is restricted by a local proxy server or firewall that is preventing a needed application from being able to connect to a server. SSH can help by allowing the application traffic to pass through a tunnel set to a port that is allowed through the firewall. Two changes would need to be made to accomplish this. First, the SSH service on the server must be modified to listen on a port known to be allowed by the firewall. A good port to use for this is port 80/TCP as most firewalls will allow outbound traffic to this port for the purposes of web browsing. And second, the "-p" parameter must be added to the command used to establish the SSH tunnel on the client followed by the custom port specified in the SSH server configuration. This parameter specifies the port on the remote SSH server that is to be used by the SSH client to connect. When the application connects, the traffic will be sent to the SSH server, past the restrictive firewall, and then forwarded to the intended destination by the SSH server.

3.3. One-time Passwords

Adds defense against: Malware, Keyloggers

One-time passwords are passwords that are generated through a mathematical algorithm, often using a random seed or the current time as a variable, and are valid for only one network session or transaction. These passwords are designed to be resistant to the major weaknesses of traditional static passwords, namely malware, network-based attacks, and brute force guessing attacks. In many cases, one-time passwords are implemented in conjunction with traditional passwords to provide an additional authentication factor.

There are two commonly used ways to implement one-time passwords, each with its own unique benefits and drawbacks. The first is to use a one-way function, usually a cryptographic hash function to create a list of one-time passwords. The process essentially works like this:

1. The cryptographic hash function is given a seed value, something specified by the user or random data generated on the computer, and hashes it to create the first key. For a simple example, let's use MD4 as the cryptographic hash function and hash the user-defined key "this is my secret password." The result is:

   1. 807902ff11d8e320d068be5264c112b6

2. The key is then hashed by the function again to create a new key and this process is repeated a pre-determined number of times. This is also known as a hash chain. Each key in the chain represents a single one-time password.

   2. c426d1250867994119744248c3dda0aa
   3. 17ec8294097f8834b0a4bcbbc1c85989
   4. 83645fb627a949ddedd78a99a696cdc3

   Since each successive key is based on the previous one, they must be used in reverse order and are numbered so that the user can be prompted for the next valid one. The user is typically presented with a complete, numbered list of the output suitable for printing. The server keeps the last key and discards the rest.

3. When a user attempts to login, he is challenged for the next to last key, 3 in this example. The value entered by the user is hashed and if it matches the value of 4 stored on the server, the user is successfully authenticated. The server then stores the value of 3 and on the next authentication request, prompts for key 2.
4. This repeats until the user runs out of one-time passwords from the list, at which point the cryptographic hash function must be run again with a new user-defined key.

This whole process works because knowing an earlier key allows you to identify a later one, but not the other way around. So an attacker that manages to capture a key as it is used is both unable to login with that same key or use the key to derive the next valid one. A weakness exists, however, in that if an attacker can obtain two sequential passwords, maybe from a user logging into the remote server twice from the same compromised machine, it is possible to conduct a brute force attack against the key.

The second common implementation of one-time passwords uses time as part of the formula for deriving passwords and has the client and server generate the same passwords independently of each other upon request instead of creating them in advance. Time-synchronized one-time passwords require either a physical hardware token containing an accurate clock that is synchronized to the authentication server, or an application on a portable device to generate and display the password. The passwords are valid only for a short span of time, usually a few minutes at most, but may be used any number of times during that period. This arguably makes time-synchronized passwords both less secure than the other implementations of one-time passwords and technically not one-time passwords at all. Because of this design choice, the authentication process may be vulnerable to network replay attacks, packet sniffing over unencrypted sessions, keyloggers and other malware. If the client is compromised with a keylogger, for example, and the attacker is fast enough (or the malware sophisticated enough), he can use the same one-time password entered by the user to authenticate to the remote resource. In fact, this issue has already been exploited by malware capable of waiting for the user to authenticate with the generated password and then acting in real-time using the password before it expires (Danchev, 2009). Nevertheless, this method is very easy

to use and support and, as a result, has been very a popular choice when deploying one-time passwords in production environments. It also does address some of the drawbacks of using cryptographic hash functions for one-time passwords in that capturing multiple passwords as they are used reveals nothing about the function used to create them. Additionally, since the passwords are created only when used, there are no lists of passwords that can be lost or stolen.

3.3.1. One-Time Passwords in Everything

Applies to: Ubuntu; anything that can be authenticated with PAM

One-Time Passwords in Everything (OPIE) is an implementation of S/key that integrates one-time password authentication with PAM (Pluggable Authentication Module). OPIE inputs a random seed, comprised of the first two characters of the host name and five random numbers, and a user-defined passphrase to a cryptographic hash function that uses MD4 or MD5 to create a hash chain. The 128-bit hexadecimal numbers output by the hashing function are first converted to 64-bit numbers and then mapped to sets of six small words taken from an English dictionary to make it easier for users to input the password. A list of passwords can be printed after creation, or if the user has access to a trusted computer, any individual password in the chain can be generated on demand using the original passphrase and seed.

The easiest way to see OPIE in action is to configure OpenSSH to utilize one-time passwords. To install OPIE, simply run the following command from a terminal window:

```
sudo apt-get install libpam-opie opie-server opie-client
```

Once installed, the PAM configuration file for SSH must be modified to make use of the new libraries. Open /etc/pam.d/sshd in an editor, go to the "# Standard Un*x authentication" section, and comment out the following line:

```
@include common-auth
```

Add these lines immediately following:

```
auth sufficient pam_unix.so
auth sufficient pam_opie.so
auth required pam_deny.so
```

With this configuration, users can choose to enter either a static password or a one-time password to authenticate. This is a good compromise for an environment where users will be logging in from secure and insecure machines. On secure machines, the user can use a traditional password and avoid the hassle of looking up the next one-time password. On untrusted machines, a one-time password can be used in lieu of the traditional password in case there is malware present. For additional security, however, both passwords can be required for all authentication attempts by simply replacing "sufficient" with "required" for pam_unix.so and pam_opie.so in the configuration file.

After configuring PAM, SSH needs to be set to allow challenge/response authentication requests. This is important, as, in order to know which one-time password to enter, the user must be prompted with the sequence number and seed of the next valid password. Edit the configuration file at /etc/ssh/sshd_config, set the "ChallengeResponseAuthentication" parameter to yes, and restart SSH.

The configuration of OPIE itself involves defining the secret passphrase and generating a list of one-time passwords with that passphrase. To start, run opiepasswd in a terminal window. When run for the first time, the program will prompt for a new secret passphrase. Once entered, the program will challenge the user for a response from a one-time password key generator, as shown below:

```
jason@Durandel:~$ opiepasswd

Adding jason:
You need the response from an OTP generator.
New secret pass phrase:
    otp-md5 499 Du4671
    Response:
```

The challenge is "otp-md5 499 Du4671" in this example. The number 499 represents the initial sequence number and also defines how many one-time passwords can be generated with this seed and passphrase, 500 in this case. The seed is shown after the sequence number. Once all of the one-time passwords have been used, opiepasswd must be run again and a new secret passphrase chosen. This must be done from a secure computer or else the secret passphrase may be compromised. So, in cases where a user will not have access to a trusted machine for a prolonged period of time, it may make sense to have a higher initial sequence number so more one-time passwords can be generated from the

same key. To do this just run the program with the "-n" command line parameter and specify the desired initial sequence number.

The output of the password generator opt-md5 will be used to answer the challenge given by opiepasswd. The otp-md5 program can be used to output one password or a range of passwords. To show the entire list of 500 passwords open a new terminal window and run otp-md5 with the "-n" parameter, which specifies the number of passwords to output. The initial sequence number and seed are also required. This will generate something similar to:

```
jason@Durandel:~$ otp-md5 -n 500 499 Du4671
Using the MD5 algorithm to compute response.
Reminder: Don't use opiekey from telnet or dial-in sessions.
Enter secret pass phrase:
0: PAD FOLK WING GIFT SNOW CHIC
...
499: SKID COIN HOFF EVEN GWYN WING
```

To output just one password, omit the "-n" parameter and replace the initial sequence number with the number of the desired password. Use the last entry in the list to answer the challenge in the first terminal window. The output should look something like:

```
jason@Durandel:~$ opiepasswd
Adding jason:
You need the response from an OTP generator.
New secret pass phrase:
    otp-md5 499 Du4671
    Response: SKID COIN HOFF EVEN GWYN WING
ID jason OTP key is 499 Du4671
SKID COIN HOFF EVEN GWYN WING
```

At this point, everything is setup and ready for use. When users connect to the machine over SSH, they will be challenged for a one-time password:

```
Daedalus:~ jason$ ssh 192.168.1.1
Password:
otp-md5 498 Du4671 ext, Response:
```

The file /etc/opiekeys contains the seed, sequence number, expected hash value, and last login time for every account configured to use one-time passwords. If a password list is ever lost, a new one can be generated with opiepasswd. Creating a new list invalidates all unused entries from a previous list, but this can only be done from a trusted machine, otherwise there is a risk that the new passphrase will be compromised. In a situation where a user is unable to get to a trusted machine, one-time password logins to a specific

account can be completely disabled by running opiepasswd with the "-d" parameter followed by the account name.

3.4. Virtual Network Computing

Defends against: Malware, File-sharing, Data left behind

Protecting against malware and file-sharing applications on public kiosks or workstations can be difficult as it is not always possible to remove or even identify such applications if they're present. This means that it is important to prevent any unauthorized applications from having access to the data being worked with. One method of accomplishing this is to have users run the necessary applications remotely on a trusted, secure computer. This has the added benefit of not leaving behind any traces of user activity on the client machine.

The best way to do this is through a remote desktop solution that allows user to connect to remote computers and interact with them as if they were physically present. There are a number of protocols and applications designed to do this, but two are especially well known and widely used: VNC and RDP.

Virtual Network Computing (VNC) displays the screen of a remote computer over a network, allowing users to interact with the computer as if they were physically present. Actions taken on the desktop are reflected on both the remote and local sides of the connection, so physical security of the VNC server should be taken into account to prevent anyone from being able to eavesdrop or gain control of the server. Some VNC products can blank the screen and lock the server so that it cannot be interacted with locally while a VNC session is active. VNC server applications are often bundled with Linux distributions, and clients exist for all major operating systems. For environments where it is not practical to use a native client application, it is possible to use web-based applications to connect to some servers.

While VNC offers protection from local threats, there are some potential issues users should be aware of. VNC relays local keyboard and mouse activity to the remote machine and transfers back an image of what is displayed on the remote screen to the local machine. This can be a bandwidth intensive process, and, on networks that are

heavily throttled or have limited connectivity, performance may be an issue. It may be possible to configure the client and server to compress the data transfer. Additionally, as this communication back and forth between the client and server occurs over TCP, it may be possible to intercept this traffic and execute a session hijacking attack or simply observe a user's activity. Some servers may provide options for encryption, but this is not a part of a standard VNC server and can break compatibility with clients. Without an additional factor for authentication – which, like encryption, is not typically supported - a keylogger on the client machine can also be used to gain unauthorized access to the remote server.

Fortunately, it is possible to secure a VNC session against keyloggers and session-related attacks by utilizing an SSH tunnel. When combined with a one-time password mechanism as discussed in the previous section, SSH can both encrypt the communication channel and provide an additional authentication factor to protect against keyloggers. The setup for this is very simple. Install and configure an SSH server to allow tunneling and require one-time passwords. Configure any firewalls to allow incoming connections to SSH. Enable the VNC server on the desired remote machine and bind it to the localhost network interface, unless the SSH server is a separate machine, in which case, configure the local firewall to allow incoming connections to this service from that machine only. With this, users will be able to establish an SSH tunnel, authenticated with a one-time password, and connect to the VNC server over the tunnel. A keylogger would still be able to compromise the user credentials used for the VNC server, but not those used to establish the tunnel. If the VNC server is configured to accept connections only from localhost (or the SSH server if separate machines) or a local firewall is configured to deny incoming requests to the VNC server, then knowledge of the credentials for the VNC server alone does not allow an attacker to connect to the server.

3.5. Remote Desktop Protocol

Defends against: Malware, File-sharing, Data left behind

Remote Desktop Protocol (RDP) was designed by Microsoft to be a Windows-centric remote desktop solution. The client application is available for most modern

operating systems and is bundled with all recent versions of Windows. It is a virtual certainty that any business workstation will have the server components to support a remote desktop session and that any public Windows computer or kiosk will have the client installed. This makes RDP an excellent choice for users looking to work with their desktop while traveling.

Remote Desktop Services, the server application that allows computers to connect for the purposes of desktop sharing, is very easily enabled in the Remote Settings tab in the System Properties dialog in Windows. Users need to also make sure that the account they wish to use is a member of the Remote Desktop Users group or an administrator. If the included Windows Firewall is enabled, an exception will be automatically enabled for RDP connections.

While RDP supports encryption natively, it is vulnerable to keyloggers just like VNC. Again though, an SSH tunnel can be used to mitigate this. The key difference in setup is that the SSH server and RDP server cannot be the same machine as the SSH server options for Windows do not support OTP authentication. The SSH server could be a virtual machine running on the RDP server (or vice versa), however. So the process would be to setup a separate SSH server that requires one-time passwords and an RDP server on a Windows workstation configured to only allow access through the firewall from the SSH server. Users would establish a port-forwarding tunnel to the SSH server and connect with a RDP client to the local host. The RDP traffic would be sent to the SSH server, which would make the connection to the RDP server on behalf of the client.

3.6. GoToMyPC

Defends against: Malware, Keyloggers, File-sharing, Data left behind

If all of the above is too complicated, or is not likely to be possible from wherever the user is traveling, a service like GoToMyPC may still be an option. GoToMyPC is a commercial remote desktop service that provides a number of security options that make it possible for users to easily and securely access a workstation from a remote site. Users start by creating an account online and installing the server application on any Mac or Windows based computers they wish to access remotely. Remote connections are

handled through a web application removing the need for a locally installed client application.

To login to a remote workstation, users first authenticate to the web application and are presented with a list of hosts that are running the server software and associated with the account. Users select a host from this list and are connected through an intermediary server that relays traffic back and forth between the client and remote host. The two are never directly connected and all traffic is encrypted with SSL. This prevents someone intercepting the traffic from determining the host the user is connecting to. At this point, users still have to authenticate to the remote workstation as well. Once logged in, the GoToMyPC application blanks the display and locks the remote computer from local interaction so anyone with physical access cannot interfere with the remote session.

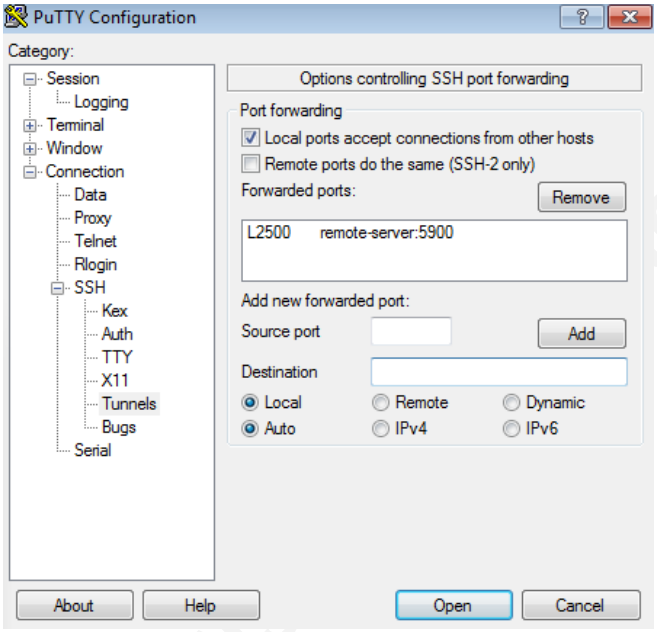
Accounts can also be protected with one-time passwords. These one-time passwords are created with a cryptographic hash function and must be generated in advance from a trusted computer. With one time passwords enabled on an account, users have to supply three unique credentials to authenticate to any remote machine, one of which is useless once used and another that is not usable unless an attacker also knows which computer was selected by the user after logging into their GoToMyPC account.

3.7. USB Devices

Defends against: Data left behind

Remote desktop applications provide great isolation and protection from threats, but may be overkill for many situations. Another way to gain some limited isolation from the host is to run applications from a USB device instead of trusting those installed on the host. This does not protect from all malware that be present on the host, but done right can prevent any relevant data about user activity from being left behind for other users to find. This works best with very simple and generic tasks, such as checking email or web browsing, where any malware may not present a threat or may be easily circumvented. Users can also take advantage of USB devices to carry applications preconfigured with security settings and user preferences with them while traveling.

In order for this to work, the applications being installed to a USB device have to be designed such that the location of all of the settings, temporary data, user profiles, and any other data needed can be configured so that nothing is written to the host. For Windows, PortableApps is a framework designed to allow applications to run entirely from USB thumb drives. There is not a lot of variety in applications, but there are applications suited to many tasks. Of particular use is a version of PuTTY, a well known SSH client for Windows. The Windows operating system does not come with an SSH client preinstalled but this version of PuTTY can be run without administrative credentials from a USB drive and used to establish SSH tunnels. The configuration dialog for SSH tunnels in PuTTY is shown in the screenshot below.



The source port refers to the local port that is to be used for the tunnel. The destination consists of the remote server and remote port parameters just like when using SSH from the command line. Multiple tunnels may be established at once through this interface. Selecting "Dynamic" instead of "Local" creates an SSH tunnel in proxy mode. With this application users can create a profile with all of the necessary SSH tunnels specified and then, when at a remote site, simply load the profile.

For more basic tasks that users may need to perform, there are PortableApps versions of Firefox, email clients, and remote desktop applications. Users can configure the version of Firefox to use SSH tunnels saved in PuTTY profiles or add the extension

FoxyProxy which allows Firefox to direct traffic over proxy servers based upon the site being visited. In effect this functions like a split VPN, only easier to setup and capable of sending traffic to multiple SSH servers. Once everything is setup, users can easily create a sandboxed environment for web browsing (or any other task) simply by running PuTTY and then Firefox with no configuration required on the remote host. The problem with this is that, except for the SSH tunnel which can still be protected with one-time passwords, any authentication required of the user is capable of being intercepted by malware on the host. There is a password management application available for PortableApps that may provide protection against basic keyloggers that don't capture screenshots or copy the clipboard. Passwords can be stored in this application, protected by a master password, and copied and pasted into authentication prompts instead of typed. Keyloggers would only compromise the master password, but without the password database stored only on the USB device, this information is not useful. Still though, users should use caution if this is the only defense being taken.

4. Conclusion

The optimal solution for any user is going to be highly dependent upon the environment at the remote site. Any information that can be ascertained about how the computers and networks are setup can be used to tailor the solution. For example, if the remote site is Mac based, then an SSH client will be available by default. Linux based workstations means that SSH VPNs may be a possibility. There are countless other solutions possible for the risks faced by users while traveling or when using untrusted or unsecured computers. Only some of the more likely usable options have been discussed at this point.

The easiest case is still that of users carrying a laptop as the laptop can be secured properly from malware and have all the necessary applications and clients installed and configured. This can be used to guarantee that certain solutions, such as VPNs, are viable and allow users to adapt to any unforeseen restrictions on network availability. The laptop serves as a secure, configurable platform for remote access. In the event that the laptop is lost, only access to the data is threatened, not the data itself, and this can be mitigated by changing passwords or disabling user accounts.

Author Name, email@addressinfosec.utexas.edu

If nothing is known about the remote site, and a laptop is not a possibility, then users have to be conservative and possibly implement multiple options to insure that some method will allow them access to their data. In this scenario, hosted remote desktop solutions such as GoToMyPC are likely the best choice. They have the benefit of being platform independent, requiring no client applications, providing encryption and security options, and allowing users access to almost any required applications. In any case, the single most important thing for a user to do is to prepare in advance as much as possible and test everything to make sure it is setup correctly before attempting to work off-site.

Author Name, email@addressinfosec.utexas.edu

5. References

Aftab, P. (2004, November 22). *The Privacy Lawyer: P2P Networks: The Other Risks*. Retrieved January 4, 2010, from InformationWeek: http://www.informationweek.com/news/security/vulnerabilities/showArticle.jhtml?articleID=53200209

Ahern, J. (2008, August 11). *Laptop Inspections Legal, Rare, Essential*. Retrieved August 17, 2209, from CBP.gov: http://www.cbp.gov/xp/cgov/travel/admissibility/labtop_inspect.xml

AirDefense. (2007, February 7). *AirDefense Discovers More Than Half of 623 Wireless Devices on Show Floor at RSA Conference Vulnerable to Attacks*. Retrieved January 6, 2010, from AirDefense: http://www.airdefense.net/newsandpress/02_07_07.php

AirDefense. (2008, January 15). *AirDefense Discovers Wireless Security Less than 'Bullet Proof' at 97th Annual National Retail Federation Convention & Expo*. Retrieved January 6, 2010, from AirDefense: http://www.airdefense.net/newsandpress/01_15_08.php

AirTight Networks. (n.d.). *Airport Vulnerability Assessment*. Retrieved January 6, 2010, from AirTight Networks: http://www.airtightnetworks.com/home/resources/knowledge-center/airport-scan.html

Baker, W. B. (2008, September). *U.S. Border Security Policy on Laptop Inspections Puts Trade Secrets at Risk*. Retrieved August 19, 2009, from Wiley Rein LLP: http://www.wileyrein.com/publications.cfm?sp=articles&id=4106

BBC. (2008, October 10). *Pension data was on stolen laptop*. Retrieved August 19, 2009, from BBC News: http://news.bbc.co.uk/2/hi/uk_news/7664274.stm

Brelsford, J. F. (n.d.). *FindLaw*. (Jones Day) Retrieved August 19, 2009, from

California Raises the Bar on Data Security and Privacy: http://library.findlaw.com/2003/Sep/30/133060.html

Chambers, C., Dolske, J., & Iyer, J. (n.d.). *TCP/IP Security*. Retrieved January 7, 2010, from LinuxSecurity: http://www.linuxsecurity.com/resource_files/documentation/tcpip-security.html

Christey, S., & Martin, R. A. (2007, May 22). *Vulnerability Type Distributions in CVE*. Retrieved January 7, 2010, from CVE: http://cve.mitre.org/docs/vuln-trends/index.html

Claburn, T. (2005, August 5). *Security Software Company Discovers Possible ID-Theft Ring*. Retrieved January 4, 2010, from InformationWeek: http://www.informationweek.com/news/security/vulnerabilities/showArticle.jhtml?articleID=167600273

Curtin, M., & Dolske, J. (1998, May). *A Brute Force Search of DES Keyspace*. Retrieved August 18, 2009, from USENIX: http://www.usenix.org/publications/login/1998-5/curtin.html

Danchev, D. (2009, September 23). *Modern banker malware undermines two-factor authentication*. Retrieved October 15, 2009, from ZDnet: http://blogs.zdnet.com/security/?p=4402

Dougherty, K. (2008, December 2). *Army waited to tell of possible security breach*. Retrieved August 19, 2009, from Stars and Stripes: https://www.stripes.com/article.asp?section=104&article=59159

Farber, D. (2007, January 16). *The secret to secure code--stop repeating old mistakes*. Retrieved January 7, 2010, from ZDNet: http://blogs.zdnet.com/BTL/?p=4280

Author Name, email@addressinfosec.utexas.edu

Fogie, S. (2005, March 11). *Cracking Wi-Fi Protected Access (WPA), Part 2*. Retrieved January 6, 2010, from Cisco: http://www.ciscopress.com/articles/article.asp?p=370636

Gong, Y. (2005, July 21). *Identifying P2P users using traffic analysis*. Retrieved September 11, 2009, from Security Focus: http://www.securityfocus.com/infocus/1843

Grebennikov, N. (2007, May 29). *Keyloggers: How they work and how to detect them (Part 1)*. Retrieved January 4, 2010, from Viruslist.com: http://www.viruslist.com/en/analysis?pubid=204791931

Hatch, B. (2004, October 14). *SSH Host Key Protection*. Retrieved January 7, 2010, from SecurityFocus: http://www.securityfocus.com/infocus/1806

Jonas, D. (2008, July 9). *Airport Laptop Seizures Debated in Washington*. Retrieved August 4, 2009, from The Transnational: http://www.thetransnational.travel/news.php?cid=laptop-seizure.Jul-08.09

Mark, R. (2005, May 05). *How Broad a Data Breach Disclosure Law?* Retrieved August 19, 2009, from internetnews.com: http://www.internetnews.com/bus-news/article.php/3502781

Mills, E. (2008, August 22). *Google making SSL changes, other sites quiet*. Retrieved January 6, 2010, from CNET News: http://news.cnet.com/8301-1009_3-10023958-83.html?tag=newsEditorsPicksArea.0

Nakashima, E. (2008, February 7). *Clarity Sought on Electronics Searches*. Retrieved August 17, 2009, from The Washington Post: http://www.washingtonpost.com/wp-dyn/content/article/2008/02/06/AR2008020604763.html

NCSA. (n.d.). *WiFi Hotspots*. Retrieved January 6, 2010, from STAYSAFEONLINE.org:

http://www.staysafeonline.org/content/wifi-hotspots

Ossmann, M. (2004, December 14). *WEP: Dead Again, Part 1*. Retrieved January 6, 2010, from Security Focus: http://www.securityfocus.com/infocus/1814

OWASP. (2009, May 27). *Session hijacking attack*. Retrieved January 6, 2010, from Open Web Application Security Project: http://www.owasp.org/index.php/Session_hijacking_attack

P2P Security. (2008, February). Retrieved January 4, 2010, from OnGuard Online: http://www.onguardonline.gov/topics/p2p-security.aspx

Poulsen, K. (2003, July 18). *Guilty Plea in Kinko's Keystroke Caper*. Retrieved August 5, 2009, from SecurityFocus: http://www.securityfocus.com/news/6447

PRWeb. (2008, September 22). *Safeware Insurance Releases Computer and Portable Electronics Loss Statistics*. Retrieved August 4, 2009, from PRWeb: http://www.prweb.com/releases/2008/09/prweb1358244.htm

Public Service. (2009, January 26). *British Council loses personal data*. Retrieved August 19, 2009, from Public Service: http://www.publicservice.co.uk/news_story.asp?id=8341

RSA Laboratories. (n.d.). *3.6.1 What is Diffie-Hellman?* Retrieved January 6, 2010, from RSA Laboratories: http://www.rsa.com/rsalabs/node.asp?id=2248

Schoultz, C. O. (2007, June 19). *Another Business Travel Concern: Laptops Being Seized at Border, Association Warns*. Retrieved August 4, 2009, from Association of Corporate Travel Executives: https://www.acte.org/resources/press_release.php?id=177

US-CERT/NIST. (2008, May 13). *Vulnerability Summary for CVE-2008-0166*. Retrieved August 18, 2009, from National Vulnerability Database: http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2008-0166

Author Name, email@addressinfosec.utexas.edu

ZapShares Inc. (2009, June 18). *New Study Shows Most People Unaware Of Security Risks Posed By P2P File-Sharing Software*. Retrieved January 4, 2010, from ZapShares: http://www.zapshares.com/pr-awareness.aspx

Author Name, email@addressinfosec.utexas.edu