**SSH:**

The SSH (Secure Shell) Protocol is a “is a cryptographic (encrypted) network protocol operating at layer 7 of the OSI Model to allow remote login and other network services to operate securely over an unsecured network”[[1]](#endnote-2). In simpler terms, the SSH protocol grants a user access to a shell (either on a remote system or local system) over a secure, encrypted network connection.

The SSH protocol uses a client-server architecture. For example, lets say a user (*UserA*) wants to access a remote workstation (*StationA*), according to the architecture, *UserA* would be the SSH client and *StationA*would be the SSH Server, although it could go the other way around as well. SSH is often used using the *ssh* command most commonly found on Unix environments.

The default protocol port is *port 22* but can be configured (along with other options) in the configuration file (/etc/ssh/sshd\_config for the server and /etc/ssh/ssh\_config for the client). Further configurations include: the type of authentication to use (Public Key or Password Authentication); the SSH protocol to use (SSHv1 or SSHv2); Permissions (which user is allowed to login); Permitted IP Addresses and plenty more options.

Because SSH is a protocol used so widely, it can expose critical information to malicious users. For example, knowing that the protocol communicates over a network, a malicious user could run a portscan (e.g using the *nmap* utility) on a company's network and identify critical infrastructure which the attack could exploit. Further more, incorrect configurations could expose private information (e.g. passwords sent over plaintext). A malicious user on the *inside* of a company could use SSH to create a tunnel and bypass certain restrictions like portals, firewalls, proxies, etc.

A company could utilize SSH into their infrastructure to access servers or workstations for remote administration. Most of the infrastructure would act as a SSH Server which a system administrator would connect to via a SSH Client like a laptop, workstation, or even a mobile device.

SSH has numerous RFCs (being that it is a protocol which has been around for a while and has undergone plenty of changes). The initial RFCs are: RFC 4250 – RFC 4256[[2]](#endnote-3), RFC 4335, 4344, 4345[[3]](#endnote-4)

There are a few major vulnerabilities for SSHv1 primarily:

* Packet injection into an SSH session.[[4]](#endnote-5)
* Arbitrary code execution via an integer overflow attack.[[5]](#endnote-6)
* Man-in-the-Middle attacks which allows the attack to re-route traffic to another malicious server using the same session id.[[6]](#endnote-7)

Due to many critical vulnerabilities in SSHv1, a new version meant to mitigate and patch these vulnerabilities was developed, SSHv2. However, SSHv2 is also susceptible to some attacks (varies based on implementation):

* Remote code execution and Denial of Service attacks.[[7]](#endnote-8)

If not configured correctly, SSH could expose multiple risks. Primarily granting accessed to unauthorized and malicious users, theft of data and intellectual property and possible denial of service attacks.

Data about SSH, such as log files and configuration files, can be found (on most \*nix systems) at:

* Configuration Files
  + /etc/ssh/ssh\_config
  + /etc/ssh/sshd\_config
  + ~/.ssh/known\_hosts
* Log Files
  + /var/log/auth
  + /var/log/secure

**WARNING:** Configuration file locations may vary from system to system, but the general idea is constant among the platforms.

Some of these risks can be mitigated by configuring SSH properly. This includes using strong Authentication methods such as Public Key Cryptography or Two Factor Authentication, allowing only authorized users to access the servers, white-listing known IP addresses and blacklisting unknown IP addresses, proper key management and firewall rule implementations (e.g. Allow only incoming and outgoing connections for certain ports, etc)

**Rsync:**

Rsync is a tool which allows a user to securely transmit files over a network (either using TCP protocol or the UDP protocol) while using the least amount of resources as possible. The protocol ensures a secure connection is made by using the SSH protocol to create a secure tunnel connection.

Rsync ensures low resources are used by using a series of checksums, one cryptographic (md5sum) to ensure the data has not been tampered with and one non-cryptographic (adler-32) uses as a Cyclic Redundancy Check to ensure there was no transmission or copy error made during the transfer. When the files on the local and remote location are not the same, Rsync will copy only the difference between the files, rather than the entire file. This process works by checksuming the file in blocks and transferring only the blocks that differ and applying the new data to the file. Ports *873/tcp* and *873/udp* are the reserved ports for the rsync *daemon.*

A business could use *rsync* to transfer files from LocationA to LocationB over a secure network. An example of this might be scheduled backups. An administrator might use *rsync* to create a backup of one machine and transfer it to another machine, a designated backup server, for example.

The *rsync* protocol only has one major RFC: RFC 5781 [[8]](#endnote-9)

There are three major Security Vulnerabilities released for the protocol:

* CVE-2007-6200[[9]](#endnote-10)
  + A vulnerability which allows an attacker to bypass file filters and read or write hidden files.
* CVE-2007-6199[[10]](#endnote-11)
  + A vulnerability which allows an attack to access unauthorized files by creating symlinks outside of the module hierarchy
* CVE-2007-4091[[11]](#endnote-12)
  + A vulnerability which allows for arbitrary code execution.

Other than the vulnerabilities directly related to rsync, an attack could exploit the underlying SSH protocol which rsync uses to establish a secure connection.

The two major risks in using the *rsync*  protocol are: 1) Loss of data if, by chance, some error occurs in the transfer of the file. This is not such a big risk because the data is being copies and not moved. This means that if some error occurs in the transmission, you can always recopy the files until the data matches, as it should. 2) If the data transferred was not encrypted and just sent in plaintext, a malicious user could sniff the traffic on the wire and steel the data being transmitted. If important files are being transferred, this may lead to a loss of intellectual property or harm to an organization.

Data files for *rsync* can be found on most \*nix systems at:

* Configuration Files
  + /etc/rsyncd.conf
* Log Files
  + rsync logs are generally sent of the syslog daemon, but only when rsync is running in daemon mode. To manually tell rsync to log the data, you can specify the –log-file on the command line when performing the rsync command.

The risks could be mitigated by ensuring you use the latest version of the rsync utility as well as the correct version of the SSH protocol.

**Gnutella:**

Gnutella is a “large peer-to-peer network. It was the first decentralized peer-to-peer network of its kind, leading to other, later networks adopting the model. It celebrated a decade of existence on March 14, 2010 and has a user base in the millions for peer-to-peer file sharing.”[[12]](#endnote-13) Essentially, Gnutella allows users to share files between peers over a network which is decentralized, that is to say, does not go through a specified central location or container centralized servers. In other words, the network is made up of individual peers connected together. The network knows which clients are on the network by utilizing Distributed Hash Tables, a system which is

“a class of a decentralized distributed system that provides a lookup service similar to a hash table: (*key*, *value*) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the nodes, in such a way that a change in the set of participants causes a minimal amount of disruption. This allows a DHT to scale to extremely large numbers of nodes and to handle continual node arrivals, departures, and failures.”[[13]](#endnote-14)

According to */etc/services*, Gnutella uses ports:

* gnutella-svc
  + 6346/tcp and 6346/udp
* gnutella-rtr
  + 6347/tcp and 6347/udp

The service might expose important information about an organizations infrastructure which an attack could use to exploit and gain access into the system. If a malicious user were to portscan a companies network and discover a Gnutella server, the attacker could use an exploit to gain entrance into the company's system and cause a denial of service, or damage such as loss of intellectual property.

A business could use the protocol internally in their company to create an isolated network on which to share data. The advantage of this is that the data is distributed across multiple peers and does not require a central client-server architecture, but rather data can be pulled from any peer currently accessible. This might be useful if the company is low on resources and cannot afford that sort of infrastructure, but can accomplish the same thing with available workstations acting as peers.

Although there does not seem to be an official IEEE documented Gnutella RFC, there is a project (RFC-Gnutella)[[14]](#endnote-15) for the purpose of documenting the evolution of the protocol.

As far as know vulnerabilities to the protocol. There is a CVE that was published:

* CVE-2001-1004[[15]](#endnote-16)
  + A cross-site scripting attack in a gnutella client.

As far as attacks on the actual architecture and implementations of the protocol. There was some research[[16]](#endnote-17) done by the University of California which describes many attacks on a Gnutella network including: Distributed Denial of Service Attacks, *Pong* Attack, IP Harvesting, Malware Injection and Privacy Violations through GUID tracing.

The major risk in using the service is users may be prone to Distributed Denial of Service attacks which could cause loss of business which could potentially harm the company.

The majority of configuration is done through a Gnutella client such as *gtk-gnutella* which allows users to configure various aspects of the system.

To mitigate the risks of gnutella, make sure to update the package and clients as frequently as possible to ensure all vulnerabilities are patched as quickly as possible. Ensure you have proper firewall rules in place to deny access to outside IP addresses that are not part of the internal company network.

1. https://en.wikipedia.org/wiki/Secure\_Shell [↑](#endnote-ref-2)
2. <https://tools.ietf.org/html/rfc4250>

   <https://tools.ietf.org/html/rfc4251>

   <https://tools.ietf.org/html/rfc4252>

   <https://tools.ietf.org/html/rfc4253>

   <https://tools.ietf.org/html/rfc4254>

   <https://tools.ietf.org/html/rfc4255>

   https://tools.ietf.org/html/rfc4256 [↑](#endnote-ref-3)
3. [https://tools.ietf.org/html/rfc4335](https://tools.ietf.org/html/rfc4355)

   <https://tools.ietf.org/html/rfc4344>

   https://tools.ietf.org/html/rfc4345 [↑](#endnote-ref-4)
4. http://www.kb.cert.org/vuls/id/13877 [↑](#endnote-ref-5)
5. http://www.kb.cert.org/vuls/id/945216 [↑](#endnote-ref-6)
6. http://www.kb.cert.org/vuls/id/684820 [↑](#endnote-ref-7)
7. <https://www.rapid7.com/resources/advisories/R7-0009.jsp>

   Rsync [↑](#endnote-ref-8)
8. https://tools.ietf.org/html/rfc5781 [↑](#endnote-ref-9)
9. https://www.cvedetails.com/cve/CVE-2007-6200/ [↑](#endnote-ref-10)
10. https://www.cvedetails.com/cve/CVE-2007-6199/ [↑](#endnote-ref-11)
11. <https://www.cvedetails.com/cve/CVE-2007-4091/>

    Gnutella [↑](#endnote-ref-12)
12. https://en.wikipedia.org/wiki/Gnutella [↑](#endnote-ref-13)
13. https://en.wikipedia.org/wiki/Distributed\_hash\_table [↑](#endnote-ref-14)
14. http://rfc-gnutella.sourceforge.net/ [↑](#endnote-ref-15)
15. https://www.cvedetails.com/cve/CVE-2001-1004/ [↑](#endnote-ref-16)
16. http://alumni.cs.ucr.edu/~csyiazti/courses/cs260-2/project/html/ [↑](#endnote-ref-17)