# Network and Internet Defences

## Computer Security Lecture 13


David Aspinall

School of Informatics
University of Edinburgh


10th March 2014

# Outline

Firewalls

Attack detection

Attack attraction

Building in security

# Outline

- Firewalls
- Attack detection
- Attack attraction
- Building in security

# Firewall varieties

Protect vulnerable machines; compensate for *impossibility* of securing internal networks.

1. **Packet filters**. Cheap, fast, stateless. Filter based on source/dest addresses, port numbers. Built into routers. Drawbacks: prevent some protocols (plain FTP, maybe UDP), dynamic port assignment (RPC).

# Firewall varieties

Protect vulnerable machines; compensate for *impossibility* of securing internal networks.

1. **Packet filters**. Cheap, fast, stateless. Filter based on source/dest addresses, port numbers. Built into routers. Drawbacks: prevent some protocols (plain FTP, maybe UDP), dynamic port assignment (RPC).
2. **Dynamic packet filters**. Stateful filters; allow more protocols by parsing command streams, portmapper messages, UDP protocols, “port knocking”. Drawback: complexity.

# Firewall varieties

Protect vulnerable machines; compensate for *impossibility* of securing internal networks.

1. **Packet filters**. Cheap, fast, stateless. Filter based on source/dest addresses, port numbers. Built into routers. Drawbacks: prevent some protocols (plain FTP, maybe UDP), dynamic port assignment (RPC).
2. **Dynamic packet filters**. Stateful filters; allow more protocols by parsing command streams, portmapper messages, UDP protocols, “port knocking”. Drawback: complexity.
3. **Application gateways**. Each app has dedicated program at firewall which acts as a relay/proxy. SMTP and HTTP work well. Drawback: gateways for each app; bottlenecks.

# Firewall varieties

Protect vulnerable machines; compensate for *impossibility* of securing internal networks.

1. **Packet filters**. Cheap, fast, stateless. Filter based on source/dest addresses, port numbers. Built into routers. Drawbacks: prevent some protocols (plain FTP, maybe UDP), dynamic port assignment (RPC).
2. **Dynamic packet filters**. Stateful filters; allow more protocols by parsing command streams, portmapper messages, UDP protocols, “port knocking”. Drawback: complexity.
3. **Application gateways**. Each app has dedicated program at firewall which acts as a relay/proxy. SMTP and HTTP work well. Drawback: gateways for each app; bottlenecks.
4. **Circuit relays**, e.g., SOCKS. Generic circuit-passing for TCP connections. Middle ground between 1 and 3. Drawbacks: poor for outgoing traffic (can even tunnel IP).

# Firewall issues

- Outbound (egress) filtering blocks launch points in DoS attacks and prevents *spyware* software which “phones home” with user information.

# Firewall issues

- Outbound (egress) filtering blocks launch points in DoS attacks and prevents *spyware* software which “phones home” with user information.
- Complex architectures use **multiple firewalls**. Outermost, a packet filter (**choke**), links to internal **demilitarized zone** (DMZ) subnet, with further app relays, filters, and isolated intranets.

# Firewall issues

- Outbound (egress) filtering blocks launch points in DoS attacks and prevents *spyware* software which "phones home" with user information.
- Complex architectures use **multiple firewalls**. Outermost, a packet filter (**choke**), links to internal **demilitarized zone** (DMZ) subnet, with further app relays, filters, and isolated intranets.
- Security cornerstone, yet **serious limitations**:

# Firewall issues

- Outbound (egress) filtering blocks launch points in DoS attacks and prevents *spyware* software which “phones home” with user information.
- Complex architectures use **multiple firewalls**. Outermost, a packet filter (**choke**), links to internal **demilitarized zone** (DMZ) subnet, with further app relays, filters, and isolated intranets.
- Security cornerstone, yet **serious limitations**:
  - Hard to configure/maintain (*tiger teams*/automated analysis).

# Firewall issues

- Outbound (egress) filtering blocks launch points in DoS attacks and prevents *spyware* software which “phones home” with user information.
- Complex architectures use **multiple firewalls**. Outermost, a packet filter (**choke**), links to internal **demilitarized zone** (DMZ) subnet, with further app relays, filters, and isolated intranets.
- Security cornerstone, yet **serious limitations**:
  - Hard to configure/maintain (*tiger teams*/automated analysis).
  - May bypass(frag’d packets, FIN-scans, **tunnels**).

# Firewall issues

- Outbound (egress) filtering blocks launch points in DoS attacks and prevents *spyware* software which “phones home” with user information.
- Complex architectures use **multiple firewalls**. Outermost, a packet filter (**choke**), links to internal **demilitarized zone** (DMZ) subnet, with further app relays, filters, and isolated intranets.
- Security cornerstone, yet **serious limitations**:
  - Hard to configure/maintain (*tiger teams*/automated analysis).
  - May bypass(frag’d packets, FIN-scans, **tunnels**).
  - Don’t prevent attacks at higher level. Circuit relay won’t prevent SMTP attacks. Application gateway may scan emails for viruses, but either accepts or rejects too much.

# Firewall issues

- Outbound (egress) filtering blocks launch points in DoS attacks and prevents *spyware* software which “phones home” with user information.
- Complex architectures use **multiple firewalls**. Outermost, a packet filter (**choke**), links to internal **demilitarized zone** (DMZ) subnet, with further app relays, filters, and isolated intranets.
- Security cornerstone, yet **serious limitations**:
  - Hard to configure/maintain (*tiger teams*/automated analysis).
  - May bypass(frag’d packets, FIN-scans, **tunnels**).
  - Don’t prevent attacks at higher level. Circuit relay won’t prevent SMTP attacks. Application gateway may scan emails for viruses, but either accepts or rejects too much.
  - Clearly can’t prevent inside attacks, or protect apps that must be exposed (web servers). Growth of web-services: “Internet interprets censorship as damage and routes around it.”

# Outline

Firewalls

Attack detection

Attack attraction

Building in security

# Logging, Auditing and Forensics

- After break-in attempts or compromise, **log files** may provide evidence and *audit trails*.

# Logging, Auditing and Forensics

- After break-in attempts or compromise, **log files** may provide evidence and *audit trails*.
- Common Unix logs (in `/var/log`): `lastlog`, `utmp` and `wtmp`, `actt` and `psacct`, `messages`, `secure`. Other programs have specific logs, e.g: `maillog`, `httpd/access_log`, `xfer_log`.

# Logging, Auditing and Forensics

- After break-in attempts or compromise, **log files** may provide evidence and *audit trails*.
- Common Unix logs (in `/var/log`): `lastlog`, `utmp` and `wtmp`, `actt` and `psacct`, `messages`, `secure`. Other programs have specific logs, e.g: `maillog`, `httpd/access_log`, `xfer_log`.
- **Beware!** If a system has been compromised, there may be no guarantee of the integrity of the log files. Countermeasures: use append only filesystem; log to a dedicated secure server or even secure printer.

# Logging, Auditing and Forensics

- After break-in attempts or compromise, **log files** may provide evidence and *audit trails*.
- Common Unix logs (in `/var/log`): `lastlog`, `utmp` and `wtmp`, `actt` and `psacct`, `messages`, `secure`. Other programs have specific logs, e.g: `maillog`, `httpd/access_log`, `xfer_log`.
- **Beware!** If a system has been compromised, there may be no guarantee of the integrity of the log files. Countermeasures: use append only filesystem; log to a dedicated secure server or even secure printer.
- Certification may require logging, but log analysis tools are limited (exceptions: swatch, logwatch).

# Logging, Auditing and Forensics

- After break-in attempts or compromise, **log files** may provide evidence and *audit trails*.
- Common Unix logs (in `/var/log`): `lastlog`, `utmp` and `wtmp`, `actt` and `psacct`, `messages`, `secure`. Other programs have specific logs, e.g: `maillog`, `httpd/access_log`, `xfer_log`.
- **Beware!** If a system has been compromised, there may be no guarantee of the integrity of the log files. Countermeasures: use append only filesystem; log to a dedicated secure server or even secure printer.
- Certification may require logging, but log analysis tools are limited (exceptions: swatch, logwatch).
- Forensics: the art of reading other less obvious, incidental trails. E.g., shell, editor, application history/lock files; secret key files; outgoing mail drops, firewall and web cache logs; ultimately file system block level or hard-drive data recovery.

# Intrusion Detection

- Realization: log and audit info was hardly used. Idea: trigger an **alarm** when some condition observed; alarm may be log/email (risks slow response) or shutdown/recovery (risks DoS).

# Intrusion Detection

- Realization: log and audit info was hardly used. Idea: trigger an **alarm** when some condition observed; alarm may be log/email (risks slow response) or shutdown/recovery (risks DoS).
  - **boundary conditions**: traditional simple tests of number of failed logins, credit card expenditure/location movement.

# Intrusion Detection

- Realization: log and audit info was hardly used. Idea: trigger an **alarm** when some condition observed; alarm may be log/email (risks slow response) or shutdown/recovery (risks DoS).
  - **boundary conditions**: traditional simple tests of number of failed logins, credit card expenditure/location movement.
  - **misuse detection**: model likely behaviour of an intruder. Scan for characteristic attack *signatures*, e.g., presence of virus, system file changes (Tripwire), execution of unusual commands, or falling into *honey trap*.

# Intrusion Detection

- Realization: log and audit info was hardly used. Idea: trigger an **alarm** when some condition observed; alarm may be log/email (risks slow response) or shutdown/recovery (risks DoS).
  - **boundary conditions**: traditional simple tests of number of failed logins, credit card expenditure/location movement.
  - **misuse detection**: model likely behaviour of an intruder. Scan for characteristic attack *signatures*, e.g., presence of virus, system file changes (Tripwire), execution of unusual commands, or falling into *honey trap*.
  - **anomaly detection**: use heuristics or neural nets to build model of normal behaviour, and then flag unusual events.

# Intrusion Detection

- Realization: log and audit info was hardly used. Idea: trigger an **alarm** when some condition observed; alarm may be log/email (risks slow response) or shutdown/recovery (risks DoS).
  - **boundary conditions**: traditional simple tests of number of failed logins, credit card expenditure/location movement.
  - **misuse detection**: model likely behaviour of an intruder. Scan for characteristic attack *signatures*, e.g., presence of virus, system file changes (Tripwire), execution of unusual commands, or falling into *honey trap*.
  - **anomaly detection**: use heuristics or neural nets to build model of normal behaviour, and then flag unusual events.
- Issues: difficult problem; Internet is noisy medium; too few attacks so more false alarms than real ones; maintaining library of attack signatures; encryption can conceal signatures.

# Outline

Firewalls

Attack detection

Attack attraction

Building in security

# Honeypots and Honeynets

- Honeypot/net: a system or network whose value lies in being probed or attacked. Not necessarily designed to attract attackers explicitly.

# Honeypots and Honeynets

- Honeypot/net: a system or network whose value lies in being probed or attacked. Not necessarily designed to attract attackers explicitly.
- Idea raised 1990/1: Clifford Stoll's book *The Cuckoo's Egg* and Bill Cheswick's paper *An Evening with Berferd*. Products appeared 1997 on.

# Honeypots and Honeynets

- Honeypot/net: a system or network whose value lies in being probed or attacked. Not necessarily designed to attract attackers explicitly.
- Idea raised 1990/1: Clifford Stoll's book *The Cuckoo's Egg* and Bill Cheswick's paper *An Evening with Berferd*. Products appeared 1997 on.
- Primary use: **gathering data** on attacks, maybe as evidence. Easy since any activity is abnormal. Standard technology: logging, packet scanning, IDS. Log security critical!

# Honeypots and Honeynets

- Honeypot/net: a system or network whose value lies in being probed or attacked. Not necessarily designed to attract attackers explicitly.
- Idea raised 1990/1: Clifford Stoll's book *The Cuckoo's Egg* and Bill Cheswick's paper *An Evening with Berferd*. Products appeared 1997 on.
- Primary use: **gathering data** on attacks, maybe as evidence. Easy since any activity is abnormal. Standard technology: logging, packet scanning, IDS. Log security critical!
- Advantages: false positives and false negatives reduced compared with IDS running on ordinary production machines. But perhaps additional risk associated in both IT and legal senses (ex: where?)

# Honeypots and Honeynets

- Honeypot/net: a system or network whose value lies in being probed or attacked. Not necessarily designed to attract attackers explicitly.
- Idea raised 1990/1: Clifford Stoll's book *The Cuckoo's Egg* and Bill Cheswick's paper *An Evening with Berferd*. Products appeared 1997 on.
- Primary use: **gathering data** on attacks, maybe as evidence. Easy since any activity is abnormal. Standard technology: logging, packet scanning, IDS. Log security critical!
- Advantages: false positives and false negatives reduced compared with IDS running on ordinary production machines. But perhaps additional risk associated in both IT and legal senses (ex: where?)
- Distinction:

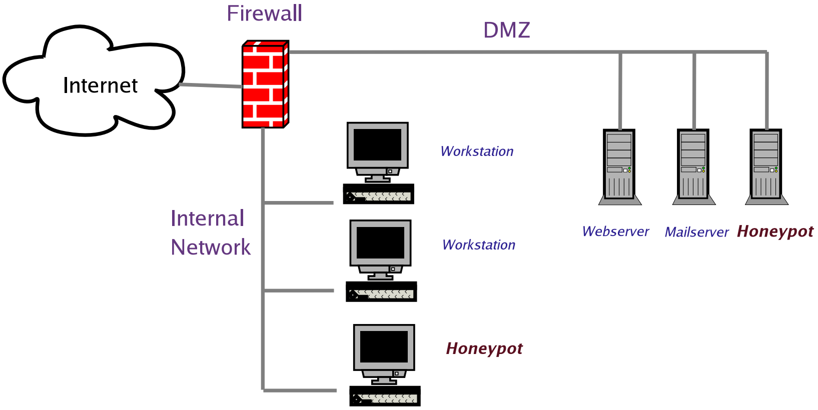
# Honeypots and Honeynets

- Honeypot/net: a system or network whose value lies in being probed or attacked. Not necessarily designed to attract attackers explicitly.
- Idea raised 1990/1: Clifford Stoll's book *The Cuckoo's Egg* and Bill Cheswick's paper *An Evening with Berferd*. Products appeared 1997 on.
- Primary use: **gathering data** on attacks, maybe as evidence. Easy since any activity is abnormal. Standard technology: logging, packet scanning, IDS. Log security critical!
- Advantages: false positives and false negatives reduced compared with IDS running on ordinary production machines. But perhaps additional risk associated in both IT and legal senses (ex: where?)
- Distinction:
  - **production honeypots**

# Honeypots and Honeynets

- Honeypot/net: a system or network whose value lies in being probed or attacked. Not necessarily designed to attract attackers explicitly.
- Idea raised 1990/1: Clifford Stoll's book *The Cuckoo's Egg* and Bill Cheswick's paper *An Evening with Berferd*. Products appeared 1997 on.
- Primary use: **gathering data** on attacks, maybe as evidence. Easy since any activity is abnormal. Standard technology: logging, packet scanning, IDS. Log security critical!
- Advantages: false positives and false negatives reduced compared with IDS running on ordinary production machines. But perhaps additional risk associated in both IT and legal senses (ex: where?)
- Distinction:
  - **production honeypots**
  - **research honeypots**

# Production Honeypot Deployment




- Production honeypots configured identically to corresponding machines. No DNS entries.

# Research Honeypots and Honeynets

- More sophisticated than production systems.

# Research Honeypots and Honeynets

- More sophisticated than production systems.
- Often a high-level of virtualization. Single machine may simulate entire heterogeneous network, including routers, workstations, printers.

# Research Honeypots and Honeynets

- More sophisticated than production systems.
- Often a high-level of virtualization. Single machine may simulate entire heterogeneous network, including routers, workstations, printers.
- Containment important: we can use **jailed environments**. For example, Unix `chroot` with customized suite of programs. Risks: attacker recognizes this, or breaks out.

# Research Honeypots and Honeynets

- More sophisticated than production systems.
- Often a high-level of virtualization. Single machine may simulate entire heterogeneous network, including routers, workstations, printers.
- Containment important: we can use **jailed environments**. For example, Unix `chroot` with customized suite of programs. Risks: attacker recognizes this, or breaks out.
- Limiting external connectivity also important: don't want to become the launch point for attacks on external networks.

# Research Honeypots and Honeynets

- More sophisticated than production systems.
- Often a high-level of virtualization. Single machine may simulate entire heterogeneous network, including routers, workstations, printers.
- Containment important: we can use **jailed environments**. For example, Unix `chroot` with customized suite of programs. Risks: attacker recognizes this, or breaks out.
- Limiting external connectivity also important: don't want to become the launch point for attacks on external networks.
- Nonetheless want to offer a high level of interaction to attackers as possible, and appear convincing (e.g. assign a domain name, fabricate a list of users, simulate network activity).

# Research Honeypots and Honeynets

- More sophisticated than production systems.
- Often a high-level of virtualization. Single machine may simulate entire heterogeneous network, including routers, workstations, printers.
- Containment important: we can use **jailed environments**. For example, Unix `chroot` with customized suite of programs. Risks: attacker recognizes this, or breaks out.
- Limiting external connectivity also important: don't want to become the launch point for attacks on external networks.
- Nonetheless want to offer a high level of interaction to attackers as possible, and appear convincing (e.g. assign a domain name, fabricate a list of users, simulate network activity).
- Advanced attackers (as opposed to script kiddies) may still be difficult to detect/attract.

# The Honeynet Project (www.honeynet.org)

- “A non-profit research organization of security professionals dedicated to learning the tools, tactics, and motives of the blackhat community and sharing the lessons learned.”

# The Honeynet Project (`www.honeynet.org`)

- “A non-profit research organization of security professionals dedicated to learning the tools, tactics, and motives of the blackhat community and sharing the lessons learned.”
- Fanciful analogy to *scouts* in military. Produced revealing series of *Know Your Enemy* papers.

# The Honeynet Project (`www.honeynet.org`)

- "A non-profit research organization of security professionals dedicated to learning the tools, tactics, and motives of the blackhat community and sharing the lessons learned."
- Fanciful analogy to *scouts* in military. Produced revealing series of *Know Your Enemy* papers.
- Started in 1999 by building (real) honeynets from standard installs of production systems. Results:

# The Honeynet Project (`www.honeynet.org`)

- "A non-profit research organization of security professionals dedicated to learning the tools, tactics, and motives of the blackhat community and sharing the lessons learned."
- Fanciful analogy to *scouts* in military. Produced revealing series of *Know Your Enemy* papers.
- Started in 1999 by building (real) honeynets from standard installs of production systems. Results:
  - End 2000: average life expectancy of standard RedHat 6.2 install was <72hrs

## The Honeynet Project (`www.honeynet.org`)

- "A non-profit research organization of security professionals dedicated to learning the tools, tactics, and motives of the blackhat community and sharing the lessons learned."
- Fanciful analogy to *scouts* in military. Produced revealing series of *Know Your Enemy* papers.
- Started in 1999 by building (real) honeynets from standard installs of production systems. Results:
  - End 2000: average life expectancy of standard RedHat 6.2 install was <72hrs
  - Records: system compromise 15mins, worm: 90secs

# The Honeynet Project (`www.honeynet.org`)

- "A non-profit research organization of security professionals dedicated to learning the tools, tactics, and motives of the blackhat community and sharing the lessons learned."
- Fanciful analogy to *scouts* in military. Produced revealing series of *Know Your Enemy* papers.
- Started in 1999 by building (real) honeynets from standard installs of production systems. Results:
  - End 2000: average life expectancy of standard RedHat 6.2 install was <72hrs
  - Records: system compromise 15mins, worm: 90secs
  - 2001: 100% increase in incidents

# The Honeynet Project (`www.honeynet.org`)

- "A non-profit research organization of security professionals dedicated to learning the tools, tactics, and motives of the blackhat community and sharing the lessons learned."
- Fanciful analogy to *scouts* in military. Produced revealing series of *Know Your Enemy* papers.
- Started in 1999 by building (real) honeynets from standard installs of production systems. Results:
  - End 2000: average life expectancy of standard RedHat 6.2 install was <72hrs
  - Records: system compromise 15mins, worm: 90secs
  - 2001: 100% increase in incidents
- CDROM *Roo*, boots into a Linux-based Honeynet gateway, or "Honeywall". Target systems placed behind the gateway; the gateway performs all Data Capture (i.e., logging) and Data Control (i.e., containment; firewalling).

# Client Honeypots (2004-)

- Realisation: honeypots mainly *passive* and watching servers, while many exploits attack clients to install malware.

# Client Honeypots (2004-)

- Realisation: honeypots mainly *passive* and watching servers, while many exploits attack clients to install malware.
- Basic idea:

# Client Honeypots (2004-)

- Realisation: honeypots mainly *passive* and watching servers, while many exploits attack clients to install malware.
- Basic idea:
  1. design a *honeyclient* that emulates or is built-on a standard client or suite (e.g., IE 6 in WinXP)

# Client Honeypots (2004-)

- Realisation: honeypots mainly *passive* and watching servers, while many exploits attack clients to install malware.
- Basic idea:
  1. design a *honeyclient* that emulates or is built-on a standard client or suite (e.g., IE 6 in WinXP)
  2. use Tripwire-like methods to monitor client, system files, registry, etc

# Client Honeypots (2004-)

- Realisation: honeypots mainly *passive* and watching servers, while many exploits attack clients to install malware.
- Basic idea:
  1. design a *honeyclient* that emulates or is built-on a standard client or suite (e.g., IE 6 in WinXP)
  2. use Tripwire-like methods to monitor client, system files, registry, etc
  3. crawl suspicious web sites or URLs in emails

# Client Honeypots (2004-)

- Realisation: honeypots mainly *passive* and watching servers, while many exploits attack clients to install malware.
- Basic idea:
  1. design a *honeyclient* that emulates or is built-on a standard client or suite (e.g., IE 6 in WinXP)
  2. use Tripwire-like methods to monitor client, system files, registry, etc
  3. crawl suspicious web sites or URLs in emails
  4. build database of malicious file alterations

# Client Honeypots (2004-)

- Realisation: honeypots mainly *passive* and watching servers, while many exploits attack clients to install malware.
- Basic idea:
  1. design a *honeyclient* that emulates or is built-on a standard client or suite (e.g., IE 6 in WinXP)
  2. use Tripwire-like methods to monitor client, system files, registry, etc
  3. crawl suspicious web sites or URLs in emails
  4. build database of malicious file alterations
  5. filter whitelist of innocuous changes

# Client Honeypots (2004-)

- Realisation: honeypots mainly *passive* and watching servers, while many exploits attack clients to install malware.
- Basic idea:
  1. design a *honeyclient* that emulates or is built-on a standard client or suite (e.g., IE 6 in WinXP)
  2. use Tripwire-like methods to monitor client, system files, registry, etc
  3. crawl suspicious web sites or URLs in emails
  4. build database of malicious file alterations
  5. filter whitelist of innocuous changes
  6. learn about exploits, build blacklist of URLs

# Client Honeypots (2004-)

- Realisation: honeypots mainly *passive* and watching servers, while many exploits attack clients to install malware.
- Basic idea:
  1. design a *honeyclient* that emulates or is built-on a standard client or suite (e.g., IE 6 in WinXP)
  2. use Tripwire-like methods to monitor client, system files, registry, etc
  3. crawl suspicious web sites or URLs in emails
  4. build database of malicious file alterations
  5. filter whitelist of innocuous changes
  6. learn about exploits, build blacklist of URLs
- Implementations:

# Client Honeypots (2004-)

- Realisation: honeypots mainly *passive* and watching servers, while many exploits attack clients to install malware.
- Basic idea:
  1. design a *honeyclient* that emulates or is built-on a standard client or suite (e.g., IE 6 in WinXP)
  2. use Tripwire-like methods to monitor client, system files, registry, etc
  3. crawl suspicious web sites or URLs in emails
  4. build database of malicious file alterations
  5. filter whitelist of innocuous changes
  6. learn about exploits, build blacklist of URLs
- Implementations:
  - MITRE **Honeyclient** (2004)

# Client Honeypots (2004-)

- Realisation: honeypots mainly *passive* and watching servers, while many exploits attack clients to install malware.
- Basic idea:
  1. design a *honeyclient* that emulates or is built-on a standard client or suite (e.g., IE 6 in WinXP)
  2. use Tripwire-like methods to monitor client, system files, registry, etc
  3. crawl suspicious web sites or URLs in emails
  4. build database of malicious file alterations
  5. filter whitelist of innocuous changes
  6. learn about exploits, build blacklist of URLs
- Implementations:
  - MITRE **Honeyclient** (2004)
  - Microsoft's **HoneyMonkey** (2005)

# Client Honeypots (2004-)

- Realisation: honeypots mainly *passive* and watching servers, while many exploits attack clients to install malware.
- Basic idea:
  1. design a *honeyclient* that emulates or is built-on a standard client or suite (e.g., IE 6 in WinXP)
  2. use Tripwire-like methods to monitor client, system files, registry, etc
  3. crawl suspicious web sites or URLs in emails
  4. build database of malicious file alterations
  5. filter whitelist of innocuous changes
  6. learn about exploits, build blacklist of URLs
- Implementations:
  - MITRE **Honeyclient** (2004)
  - Microsoft's **HoneyMonkey** (2005)
  - **Google Safe Browsing API** (2008)

# Client Honeypots (2004-)

- Realisation: honeypots mainly *passive* and watching servers, while many exploits attack clients to install malware.
- Basic idea:
  1. design a *honeyclient* that emulates or is built-on a standard client or suite (e.g., IE 6 in WinXP)
  2. use Tripwire-like methods to monitor client, system files, registry, etc
  3. crawl suspicious web sites or URLs in emails
  4. build database of malicious file alterations
  5. filter whitelist of innocuous changes
  6. learn about exploits, build blacklist of URLs
- Implementations:
  - MITRE **Honeyclient** (2004)
  - Microsoft's **HoneyMonkey** (2005)
  - **Google Safe Browsing API** (2008)
- Again, needs carefully designed resilient architecture.

# Outline

- Firewalls
- Attack detection
- Attack attraction
- Building in security

# Securing Unsecured Networks

- **Link-level security**. Confidentiality and authentication ensured on individual links.

# Securing Unsecured Networks

- **Link-level security**. Confidentiality and authentication ensured on individual links.
  - Most transparent; implemented by low-level hardware.

# Securing Unsecured Networks

- **Link-level security**. Confidentiality and authentication ensured on individual links.
  - Most transparent; implemented by low-level hardware.
  - Appropriate only for local traffic, or small number of vulnerable lines.

# Securing Unsecured Networks

- **Link-level security**. Confidentiality and authentication ensured on individual links.
  - Most transparent; implemented by low-level hardware.
  - Appropriate only for local traffic, or small number of vulnerable lines.
  - Examples: satellite circuits, transatlantic cables, and Wi-Fi Protected Access (**WPA**).

# Securing Unsecured Networks

- **Link-level security**. Confidentiality and authentication ensured on individual links.
  - Most transparent; implemented by low-level hardware.
  - Appropriate only for local traffic, or small number of vulnerable lines.
  - Examples: satellite circuits, transatlantic cables, and Wi-Fi Protected Access (**WPA**).

- **Network/transport-level secruity**. Conversations secured in the networking protocol.

# Securing Unsecured Networks

- **Link-level security**. Confidentiality and authentication ensured on individual links.
  - Most transparent; implemented by low-level hardware.
  - Appropriate only for local traffic, or small number of vulnerable lines.
  - Examples: satellite circuits, transatlantic cables, and Wi-Fi Protected Access (**WPA**).

- **Network/transport-level secruity**. Conversations secured in the networking protocol.
  - Transparent to applications, but can set security needs by need and negotiation.

# Securing Unsecured Networks

- **Link-level security**. Confidentiality and authentication ensured on individual links.
  - Most transparent; implemented by low-level hardware.
  - Appropriate only for local traffic, or small number of vulnerable lines.
  - Examples: satellite circuits, transatlantic cables, and Wi-Fi Protected Access (**WPA**).
- **Network/transport-level secruity**. Conversations secured in the networking protocol.
  - Transparent to applications, but can set security needs by need and negotiation.
  - E.g., for the Internet, **IPsec**.

# Securing Unsecured Networks

- **Application-level security**. Confidentiality and authentication secured by the application.

# Securing Unsecured Networks

- **Application-level security**. Confidentiality and authentication secured by the application.
  - Least convenient (each app must be modified)

# Securing Unsecured Networks

- **Application-level security**. Confidentiality and authentication secured by the application.
  - Least convenient (each app must be modified)
  - . . . but most flexible: can be customized for application concerned

# Securing Unsecured Networks

- **Application-level security**. Confidentiality and authentication secured by the application.
  - Least convenient (each app must be modified)
  - . . . but most flexible: can be customized for application concerned
  - Examples include **ssh** for remote login, **SSL/TLS** designed for secure web transactions, and **S/MIME** or **PGP** for secured email.

# IPsec and IPv6

- **IPv6** adds strong crypto security services to IP. **IPsec** is the retrofit to IPv4. Three mechanisms:

# IPsec and IPv6

- **IPv6** adds strong crypto security services to IP. **IPsec** is the retrofit to IPv4. Three mechanisms:
- **Authentication Header (AH)** [RFC2402]

# IPsec and IPv6

- **IPv6** adds strong crypto security services to IP. **IPsec** is the retrofit to IPv4. Three mechanisms:
- **Authentication Header (AH)** [RFC2402]
  - New header after the IP header used for authentication.

# IPsec and IPv6

- **IPv6** adds strong crypto security services to IP. **IPsec** is the retrofit to IPv4. Three mechanisms:
- **Authentication Header (AH)** [RFC2402]
  - New header after the IP header used for authentication.
  - Includes *SPI*; sequence no; integrity check hash.

# IPsec and IPv6

- **IPv6** adds strong crypto security services to IP. **IPsec** is the retrofit to IPv4. Three mechanisms:
- **Authentication Header (AH)** [RFC2402]
  - New header after the IP header used for authentication.
  - Includes *SPI*; sequence no; integrity check hash.
- **Encapsulating Security Payload (ESP)** [RFC2406]

# IPsec and IPv6

- **IPv6** adds strong crypto security services to IP. **IPsec** is the retrofit to IPv4. Three mechanisms:
- **Authentication Header (AH)** [RFC2402]
  - New header after the IP header used for authentication.
  - Includes *SPI*; sequence no; integrity check hash.
- **Encapsulating Security Payload (ESP)** [RFC2406]
  - Encryption mechanism providing confidentiality and/or authentication. (Originally purely confidentiality, but then attacks were discovered).

# IPsec and IPv6

- **IPv6** adds strong crypto security services to IP. **IPsec** is the retrofit to IPv4. Three mechanisms:
- **Authentication Header (AH)** [RFC2402]
  - New header after the IP header used for authentication.
  - Includes *SPI*; sequence no; integrity check hash.
- **Encapsulating Security Payload (ESP)** [RFC2406]
  - Encryption mechanism providing confidentiality and/or authentication. (Originally purely confidentiality, but then attacks were discovered).
- **Internet Key Exchange protocol (IKE)** [RFC2409]

# IPsec and IPv6

- **IPv6** adds strong crypto security services to IP. **IPsec** is the retrofit to IPv4. Three mechanisms:
- **Authentication Header (AH)** [RFC2402]
  - New header after the IP header used for authentication.
  - Includes *SPI*; sequence no; integrity check hash.
- **Encapsulating Security Payload (ESP)** [RFC2406]
  - Encryption mechanism providing confidentiality and/or authentication. (Originally purely confidentiality, but then attacks were discovered).
- **Internet Key Exchange protocol (IKE)** [RFC2409]
  - Protocol for negotiating security and authentication/encryption keys

# IPsec and IPv6

- **IPv6** adds strong crypto security services to IP. **IPsec** is the retrofit to IPv4. Three mechanisms:
- **Authentication Header (AH)** [RFC2402]
  - New header after the IP header used for authentication.
  - Includes *SPI*; sequence no; integrity check hash.
- **Encapsulating Security Payload (ESP)** [RFC2406]
  - Encryption mechanism providing confidentiality and/or authentication. (Originally purely confidentiality, but then attacks were discovered).
- **Internet Key Exchange protocol (IKE)** [RFC2409]
  - Protocol for negotiating security and authentication/encryption keys
  - Uses Diffie-Hellman (i.e., key agreement of fresh shared key without authentication).

# IPsec: Security Associations

- The **Internet Security Association and Key Management Protocol (ISAKMP)** [RFC2408], describes negotiating a *security association* (SA), which defines:

# IPsec: Security Associations

- The **Internet Security Association and Key Management Protocol (ISAKMP)** [RFC2408], describes negotiating a *security association* (SA), which defines:
  1. a destination IP,

# IPsec: Security Associations

- The **Internet Security Association and Key Management Protocol (ISAKMP)** [RFC2408], describes negotiating a *security association* (SA), which defines:
  1. a destination IP,
  2. a protocol ID,

# IPsec: Security Associations

- The **Internet Security Association and Key Management Protocol (ISAKMP)** [RFC2408], describes negotiating a *security association* (SA), which defines:
  1. a destination IP,
  2. a protocol ID,
  3. an SPI (*security parameter index*), an identifier to track SAs.

# IPsec: Security Associations

- The **Internet Security Association and Key Management Protocol (ISAKMP)** [RFC2408], describes negotiating a *security association* (SA), which defines:
  1. a destination IP,
  2. a protocol ID,
  3. an SPI (*security parameter index*), an identifier to track SAs.
- Security association meaningful for destination end only: peer-to-peer security requires two SAs.

# IPsec: Security Associations

- The **Internet Security Association and Key Management Protocol (ISAKMP)** [RFC2408], describes negotiating a *security association* (SA), which defines:
  1. a destination IP,
  2. a protocol ID,
  3. an SPI (*security parameter index*), an identifier to track SAs.
- Security association meaningful for destination end only: peer-to-peer security requires two SAs.
- SAs are usually negotiated dynamically using IKE, although other protocols possible.

# IPsec: Security Associations

- The **Internet Security Association and Key Management Protocol (ISAKMP)** [RFC2408], describes negotiating a *security association* (SA), which defines:
  1. a destination IP,
  2. a protocol ID,
  3. an SPI (*security parameter index*), an identifier to track SAs.
- Security association meaningful for destination end only: peer-to-peer security requires two SAs.
- SAs are usually negotiated dynamically using IKE, although other protocols possible.
- IKE is rather complicated (allows for extending SAs, deleting SAs, detecting dead peers), which has raised interoperability problems. A Kerberos-based protocol and simplified version, IKEv2 (2005), may replace it.

# IPsec: AH and ESP

- To use AH with an IPv6/IPsec datagram, the sender:

# IPsec: AH and ESP

- To use AH with an IPv6/IPsec datagram, the sender:
  - locates a SA to determine the mechanism

# IPsec: AH and ESP

- To use AH with an IPv6/IPsec datagram, the sender:
  - locates a SA to determine the mechanism
  - calculates the authentication data based on the ready part of the packet (uninitialized fields, e.g., authentication data, are zeroed).

# IPsec: AH and ESP

- To use AH with an IPv6/IPsec datagram, the sender:
  - locates a SA to determine the mechanism
  - calculates the authentication data based on the ready part of the packet (uninitialized fields, e.g., authentication data, are zeroed).

# IPsec: AH and ESP

- To use AH with an IPv6/IPsec datagram, the sender:
  - locates a SA to determine the mechanism
  - calculates the authentication data based on the ready part of the packet (uninitialized fields, e.g., authentication data, are zeroed).

  A MAC such as HMAC with MD5, SHA-1 is used.
- Similarly, to use ESP with an IPv6/IPsec datagram, the sender:

# IPsec: AH and ESP

- To use AH with an IPv6/IPsec datagram, the sender:
  - locates a SA to determine the mechanism
  - calculates the authentication data based on the ready part of the packet (uninitialized fields, e.g., authentication data, are zeroed).

  A MAC such as HMAC with MD5, SHA-1 is used.
- Similarly, to use ESP with an IPv6/IPsec datagram, the sender:
  - locates a SA to determine the mechanism

# IPsec: AH and ESP

- To use AH with an IPv6/IPsec datagram, the sender:
  - locates a SA to determine the mechanism
  - calculates the authentication data based on the ready part of the packet (uninitialized fields, e.g., authentication data, are zeroed).

  A MAC such as HMAC with MD5, SHA-1 is used.
- Similarly, to use ESP with an IPv6/IPsec datagram, the sender:
  - locates a SA to determine the mechanism
  - calculates the encryption and/or authentication

# IPsec: AH and ESP

- To use AH with an IPv6/IPsec datagram, the sender:
  - locates a SA to determine the mechanism
  - calculates the authentication data based on the ready part of the packet (uninitialized fields, e.g., authentication data, are zeroed).

  A MAC such as HMAC with MD5, SHA-1 is used.
- Similarly, to use ESP with an IPv6/IPsec datagram, the sender:
  - locates a SA to determine the mechanism
  - calculates the encryption and/or authentication
- There is much flexibility over where IPsec is placed: encryption may occur at hosts or routers; packets may be sent in a *transport* or *tunneled* mode.

# IPsec in Transport mode

- In transport mode, the AH is inserted after the IP header and before an upper layer protocol (e.g., TCP, UDP, ICMP).

# IPsec in Transport mode

- In transport mode, the AH is inserted after the IP header and before an upper layer protocol (e.g., TCP, UDP, ICMP).
- Original IPv4 packet:

# IPsec in Transport mode

- In transport mode, the AH is inserted after the IP header and before an upper layer protocol (e.g., TCP, UDP, ICMP).
- Original IPv4 packet:

| | | |
|---|---|---|
| *IP hdr* | *TCP hdr* | *User Data* |

# IPsec in Transport mode

- In transport mode, the AH is inserted after the IP header and before an upper layer protocol (e.g., TCP, UDP, ICMP).
- Original IPv4 packet:

| | | |
|---|---|---|
| *IP hdr* | *TCP hdr* | *User Data* |

# IPsec in Transport mode

- In transport mode, the AH is inserted after the IP header and before an upper layer protocol (e.g., TCP, UDP, ICMP).
- Original IPv4 packet:

| | | |
|---|---|---|
| *IP hdr* | *TCP hdr* | *User Data* |

becomes:

| | | | |
|---|---|---|---|
| *IP hdr* | *AH hdr* | *TCP hdr* | *User Data* |

|←— *authenticated* —→|

# IPsec in Transport mode

- In transport mode, the AH is inserted after the IP header and before an upper layer protocol (e.g., TCP, UDP, ICMP).
- Original IPv4 packet:

| | | |
|---|---|---|
| *IP hdr* | *TCP hdr* | *User Data* |

becomes:

| | | | |
|---|---|---|---|
| *IP hdr* | *AH hdr* | *TCP hdr* | *User Data* |

|⟵ *authenticated* ⟶|

- Authentication doesn't apply to mutable fields of IP header.

# IPsec in Transport mode

- In transport mode, the AH is inserted after the IP header and before an upper layer protocol (e.g., TCP, UDP, ICMP).
- Original IPv4 packet:

| *IP hdr* | *TCP hdr* | *User Data* |
|---|---|---|

becomes:

| *IP hdr* | *AH hdr* | *TCP hdr* | *User Data* |
|---|---|---|---|

|⟵ *authenticated* ⟶|

- Authentication doesn't apply to mutable fields of IP header.
- ESP in transport mode similar, except a trailer is added to user data (including encryption padding) before encrypting. Encryption applies to TCP header, user data, and trailer. Authentication field is added at the end. Minor difference: no authentication of IP header.

# IPsec in Tunnel mode

- In tunnel mode, the “inner” IP header carries the ultimate source and destination addresses, whereas an “outer” IP header may contain other addresses, e.g., addresses of security gateways.

# IPsec in Tunnel mode

- In tunnel mode, the “inner” IP header carries the ultimate source and destination addresses, whereas an “outer” IP header may contain other addresses, e.g., addresses of security gateways.
- An IPv4 packet before:

# IPsec in Tunnel mode

- In tunnel mode, the “inner” IP header carries the ultimate source and destination addresses, whereas an “outer” IP header may contain other addresses, e.g., addresses of security gateways.
- An IPv4 packet before:

| | | |
|---|---|---|
| *IP hdr* | *TCP hdr* | *User Data* |

# IPsec in Tunnel mode

- In tunnel mode, the “inner” IP header carries the ultimate source and destination addresses, whereas an “outer” IP header may contain other addresses, e.g., addresses of security gateways.
- An IPv4 packet before:

| | | |
|---|---|---|
| *IP hdr* | *TCP hdr* | *User Data* |

# IPsec in Tunnel mode

- In tunnel mode, the “inner” IP header carries the ultimate source and destination addresses, whereas an “outer” IP header may contain other addresses, e.g., addresses of security gateways.
- An IPv4 packet before:

| | | |
|---|---|---|
| *IP hdr* | *TCP hdr* | *User Data* |

and after:

| | | | | |
|---|---|---|---|---|
| *new IP hdr* | *AH hdr* | *old IP hdr* | *TCP hdr* | *User Data* |

|⟵ *authenticated* ⟶|

# IPsec in Tunnel mode

- In tunnel mode, the "inner" IP header carries the ultimate source and destination addresses, whereas an "outer" IP header may contain other addresses, e.g., addresses of security gateways.
- An IPv4 packet before:

| | | |
|---|---|---|
| *IP hdr* | *TCP hdr* | *User Data* |

  and after:

| | | | | |
|---|---|---|---|---|
| *new IP hdr* | *AH hdr* | *old IP hdr* | *TCP hdr* | *User Data* |

|←— *authenticated* —→|

- Authentication doesn't apply to mutable fields of new IP header.

# IPsec in Tunnel mode

- In tunnel mode, the “inner” IP header carries the ultimate source and destination addresses, whereas an “outer” IP header may contain other addresses, e.g., addresses of security gateways.
- An IPv4 packet before:

| *IP hdr* | *TCP hdr* | *User Data* |
|---|---|---|

  and after:

| *new IP hdr* | *AH hdr* | *old IP hdr* | *TCP hdr* | *User Data* |
|---|---|---|---|---|

|←— *authenticated* —→|

- Authentication doesn’t apply to mutable fields of new IP header.
- ESP in tunnel mode encrypts the original IP header, TCP header, user data, and the ESP trailer (padding). An extra authentication field is appended. Again, authentication of the new IP header is omitted with ESP.

# IPsec: summary

- Advantages:

# IPsec: summary

- Advantages:
  - provides security transparently for all applications;

# IPsec: summary

- Advantages:
  - provides security transparently for all applications;
  - adds to IP level end-to-end data reliability, secure sequencing of datagrams, authentication and confidentiality;

# IPsec: summary

- Advantages:
  - provides security transparently for all applications;
  - adds to IP level end-to-end data reliability, secure sequencing of datagrams, authentication and confidentiality;
  - in long term, likely to improve overall Internet infrastructure and security.

# IPsec: summary

- Advantages:
  - provides security transparently for all applications;
  - adds to IP level end-to-end data reliability, secure sequencing of datagrams, authentication and confidentiality;
  - in long term, likely to improve overall Internet infrastructure and security.
- Disadvantages

# IPsec: summary

- Advantages:
  - provides security transparently for all applications;
  - adds to IP level end-to-end data reliability, secure sequencing of datagrams, authentication and confidentiality;
  - in long term, likely to improve overall Internet infrastructure and security.
- Disadvantages
  - crypto operations impinge on throughput and latency everywhere, irrespective of security needs;

# IPsec: summary

- Advantages:
  - provides security transparently for all applications;
  - adds to IP level end-to-end data reliability, secure sequencing of datagrams, authentication and confidentiality;
  - in long term, likely to improve overall Internet infrastructure and security.
- Disadvantages
  - crypto operations impinge on throughput and latency everywhere, irrespective of security needs;
  - security model is low-level and may be disconnected from application level (e.g., authentication is host-based, not user-based);

# IPsec: summary

- Advantages:
  - provides security transparently for all applications;
  - adds to IP level end-to-end data reliability, secure sequencing of datagrams, authentication and confidentiality;
  - in long term, likely to improve overall Internet infrastructure and security.
- Disadvantages
  - crypto operations impinge on throughput and latency everywhere, irrespective of security needs;
  - security model is low-level and may be disconnected from application level (e.g., authentication is host-based, not user-based);
  - complex to implement, choice of configurations;

# IPsec: summary

- Advantages:
  - provides security transparently for all applications;
  - adds to IP level end-to-end data reliability, secure sequencing of datagrams, authentication and confidentiality;
  - in long term, likely to improve overall Internet infrastructure and security.
- Disadvantages
  - crypto operations impinge on throughput and latency everywhere, irrespective of security needs;
  - security model is low-level and may be disconnected from application level (e.g., authentication is host-based, not user-based);
  - complex to implement, choice of configurations;
  - does not prevent traffic analysis or covert channels.

# DNS Security

- DNS Security design dates back to 1993; deployment increasing now. DNS data (RRsets, *Resource Record sets*) is considered public, so no confidentiality provision; security mechanisms add authentication and integrity by digital signatures.

# DNS Security

- DNS Security design dates back to 1993; deployment increasing now. DNS data (RRsets, *Resource Record sets*) is considered public, so no confidentiality provision; security mechanisms add authentication and integrity by digital signatures.
- The DNSSEC extensions provide three services:

# DNS Security

- DNS Security design dates back to 1993; deployment increasing now. DNS data (RRsets, *Resource Record sets*) is considered public, so no confidentiality provision; security mechanisms add authentication and integrity by digital signatures.
- The DNSSEC extensions provide three services:
  1. **data origin authentication and integrity**, using public *zone keys*. Security-aware resolvers build a chain of trust.

# DNS Security

- DNS Security design dates back to 1993; deployment increasing now. DNS data (RRsets, *Resource Record sets*) is considered public, so no confidentiality provision; security mechanisms add authentication and integrity by digital signatures.
- The DNSSEC extensions provide three services:
  1. **data origin authentication and integrity**, using public *zone keys*. Security-aware resolvers build a chain of trust.
  2. **key distribution**, so servers transmit keys

# DNS Security

- DNS Security design dates back to 1993; deployment increasing now. DNS data (RRsets, *Resource Record sets*) is considered public, so no confidentiality provision; security mechanisms add authentication and integrity by digital signatures.
- The DNSSEC extensions provide three services:
  1. **data origin authentication and integrity**, using public *zone keys*. Security-aware resolvers build a chain of trust.
  2. **key distribution**, so servers transmit keys
  3. **transaction and request authentication** for DNS.

# DNS Security

- DNS Security design dates back to 1993; deployment increasing now. DNS data (RRsets, *Resource Record sets*) is considered public, so no confidentiality provision; security mechanisms add authentication and integrity by digital signatures.
- The DNSSEC extensions provide three services:
  1. **data origin authentication and integrity**, using public *zone keys*. Security-aware resolvers build a chain of trust.
  2. **key distribution**, so servers transmit keys
  3. **transaction and request authentication** for DNS.
- New security-related RRs are added:

# DNS Security

- DNS Security design dates back to 1993; deployment increasing now. DNS data (RRsets, *Resource Record sets*) is considered public, so no confidentiality provision; security mechanisms add authentication and integrity by digital signatures.
- The DNSSEC extensions provide three services:
  1. **data origin authentication and integrity**, using public *zone keys*. Security-aware resolvers build a chain of trust.
  2. **key distribution**, so servers transmit keys
  3. **transaction and request authentication** for DNS.
- New security-related RRs are added:
  - KEY record, for public keys (specifying algorithm)

# DNS Security

- DNS Security design dates back to 1993; deployment increasing now. DNS data (RRsets, *Resource Record sets*) is considered public, so no confidentiality provision; security mechanisms add authentication and integrity by digital signatures.
- The DNSSEC extensions provide three services:
  1. **data origin authentication and integrity**, using public *zone keys*. Security-aware resolvers build a chain of trust.
  2. **key distribution**, so servers transmit keys
  3. **transaction and request authentication** for DNS.
- New security-related RRs are added:
  - KEY record, for public keys (specifying algorithm)
  - SIG record, for attaching digital signatures

# DNS Security

- DNS Security design dates back to 1993; deployment increasing now. DNS data (RRsets, *Resource Record sets*) is considered public, so no confidentiality provision; security mechanisms add authentication and integrity by digital signatures.
- The DNSSEC extensions provide three services:
  1. **data origin authentication and integrity**, using public *zone keys*. Security-aware resolvers build a chain of trust.
  2. **key distribution**, so servers transmit keys
  3. **transaction and request authentication** for DNS.
- New security-related RRs are added:
  - `KEY` record, for public keys (specifying algorithm)
  - `SIG` record, for attaching digital signatures
  - `NXT` record, for *non existence*. Secure negative responses.

# DNS Security

- DNS Security design dates back to 1993; deployment increasing now. DNS data (RRsets, *Resource Record sets*) is considered public, so no confidentiality provision; security mechanisms add authentication and integrity by digital signatures.
- The DNSSEC extensions provide three services:
  1. **data origin authentication and integrity**, using public *zone keys*. Security-aware resolvers build a chain of trust.
  2. **key distribution**, so servers transmit keys
  3. **transaction and request authentication** for DNS.
- New security-related RRs are added:
  - `KEY` record, for public keys (specifying algorithm)
  - `SIG` record, for attaching digital signatures
  - `NXT` record, for *non existence*. Secure negative responses.
- Many further issues (caching, insecure compatibility, etc).

# The Secure Shell

- **SSH** is a set of programs that offers secure TCP communications between two systems, regardless of untrusted systems between them (routers, firewalls, sniffers, etc.). A powerful security tool.

# The Secure Shell

- **SSH** is a set of programs that offers secure TCP communications between two systems, regardless of untrusted systems between them (routers, firewalls, sniffers, etc.). A powerful security tool.
- Provides secure replacements for `telnet`, `rsh`, `rcp`, `rlogin`, `ftp`. Can be a secure tunnel for any TCP service; a cheap VPN-alike (e.g., ppp over ssh).

# The Secure Shell

- **SSH** is a set of programs that offers secure TCP communications between two systems, regardless of untrusted systems between them (routers, firewalls, sniffers, etc.). A powerful security tool.
- Provides secure replacements for `telnet`, `rsh`, `rcp`, `rlogin`, `ftp`. Can be a secure tunnel for any TCP service; a cheap VPN-alike (e.g., ppp over ssh).
- Offers encryption, authentication, integrity. Protects against IP and DNS spoofing, fake routes, MITM, replay.

# The Secure Shell

- **SSH** is a set of programs that offers secure TCP communications between two systems, regardless of untrusted systems between them (routers, firewalls, sniffers, etc.). A powerful security tool.
- Provides secure replacements for `telnet`, `rsh`, `rcp`, `rlogin`, `ftp`. Can be a secure tunnel for any TCP service; a cheap VPN-alike (e.g., ppp over ssh).
- Offers encryption, authentication, integrity. Protects against IP and DNS spoofing, fake routes, MITM, replay.
- Flexible choice of ciphers. Implementations for various platforms, including free OpenSSH.

# The Secure Shell

- **SSH** is a set of programs that offers secure TCP communications between two systems, regardless of untrusted systems between them (routers, firewalls, sniffers, etc.). A powerful security tool.
- Provides secure replacements for `telnet`, `rsh`, `rcp`, `rlogin`, `ftp`. Can be a secure tunnel for any TCP service; a cheap VPN-alike (e.g., ppp over ssh).
- Offers encryption, authentication, integrity. Protects against IP and DNS spoofing, fake routes, MITM, replay.
- Flexible choice of ciphers. Implementations for various platforms, including free OpenSSH.
- Disadvantages: need to carry private key around; still vulnerable to DoS attacks (connection terminations) by injected IP packets.

# Virtual Private Networks

- Extend the boundary of a protected domain, e.g. for:

# Virtual Private Networks

- Extend the boundary of a protected domain, e.g. for:
  - Remote branch offices or business collaborations. Shared file systems, logins, databases.

# Virtual Private Networks

- Extend the boundary of a protected domain, e.g. for:
  - Remote branch offices or business collaborations. Shared file systems, logins, databases.
  - Telecommuting. Tricky issues over IP addresses, routing and DNS.

# Virtual Private Networks

- Extend the boundary of a protected domain, e.g. for:
  - Remote branch offices or business collaborations. Shared file systems, logins, databases.
  - Telecommuting. Tricky issues over IP addresses, routing and DNS.

- Implementations in software or hardware

# Virtual Private Networks

- Extend the boundary of a protected domain, e.g. for:
  - Remote branch offices or business collaborations. Shared file systems, logins, databases.
  - Telecommuting. Tricky issues over IP addresses, routing and DNS.

- Implementations in software or hardware
  - Software: pros: configurability; cons: complexity, compromises.

# Virtual Private Networks

- Extend the boundary of a protected domain, e.g. for:
  - Remote branch offices or business collaborations. Shared file systems, logins, databases.
  - Telecommuting. Tricky issues over IP addresses, routing and DNS.
- Implementations in software or hardware
  - Software: pros: configurability; cons: complexity, compromises.
  - Hardware: pros: simplicity

# Virtual Private Networks

- Extend the boundary of a protected domain, e.g. for:
  - Remote branch offices or business collaborations. Shared file systems, logins, databases.
  - Telecommuting. Tricky issues over IP addresses, routing and DNS.

- Implementations in software or hardware
  - Software: pros: configurability; cons: complexity, compromises.
  - Hardware: pros: simplicity

- Security by encapsulation in the network level, using e.g. IPsec, L2TPv3+IPsec, SSL/TLS.

# Other defences, mechanisms and tools

- **Kerberos**: secure authentication system for networks: *tickets* with short lifetimes, reduces password traffic on network. Applications have to be adapted to use Kerberos libraries. Improves security inside network perimeters (compared with host-based trust on network services).

# Other defences, mechanisms and tools

- **Kerberos**: secure authentication system for networks: *tickets* with short lifetimes, reduces password traffic on network. Applications have to be adapted to use Kerberos libraries. Improves security inside network perimeters (compared with host-based trust on network services).
- **SRP**, *Secure Remote Password* is an authentication protocol which avoids encryption algorithms, allows short passwords, and stores sensitive information on server so that it cannot be subjected to dictionary attack.

# Other defences, mechanisms and tools

- **Kerberos**: secure authentication system for networks: *tickets* with short lifetimes, reduces password traffic on network. Applications have to be adapted to use Kerberos libraries. Improves security inside network perimeters (compared with host-based trust on network services).
- **SRP**, *Secure Remote Password* is an authentication protocol which avoids encryption algorithms, allows short passwords, and stores sensitive information on server so that it cannot be subjected to dictionary attack.
- SSL/TLS-enhanced protocols e.g., **SSLtelnet**, **SSLftp**, **stunnel**.

# References


- Edward G. Amoroso. *Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Trace Back, Traps and Response*. Intrusion.Net, 1999.
- Simson Garfinkel, Gene Spafford, and Alan Schwartz. *Practical UNIX and Internet Security*. O'Reilly, 3rd edition, 2003.
- Lance Spitzner. *Honeypots: tracking hackers*. Addison-Wesley, 2003.
- William R Cheswick, Steven M Bellovin, and Aviel D Rubin. *Firewalls and Internet Security Second Edition: Repelling the Wily Hacker*. Addison-Wesley, 2003.


**Recommended Reading**

Part II of Cheswick (1st edition available online).