System Security, part 2

ComS 252 — Iowa State University

Andrew Miner and Barry Britt

Disclaimer

- I am not a security expert
- This class cannot make you a security expert
  - Two lectures are not nearly enough
  - The prereqs for this class are not enough
- I will teach you
  - What you are up against (i.e., why this is hard)
  - General principles to make your system safer
  - Utilities that can help you

What is “authorization”

- Authorization refers to who is allowed to do what
- Based on userIDs, groupIDs, and permissions
- The ability to do something is usually referred to as privilege
  - E.g., “do you have sufficient privileges to read that file”
  - Note that privileges may be granted or revoked
- The `root` account has sufficient privileges to do anything
  - Except perhaps for things that are not “do-able”

Principle of Least Privilege

This is such a fundamental philosophy in system security that it gets its own slide

Principle of Least Privilege

This is such a fundamental philosophy in system security that it gets its own slide

The principle of least privilege

Every entity (user, process, or program) must be able to access only the resources necessary for its legitimate purpose.

Consequences of least privilege

- There will be several “system” accounts
  - Daemons that are not required to run as `root`, shouldn’t
  - So we get various accounts for various daemons
    - E.g., user `apache` for running `httpd`
  - These accounts do not have login shells

- You should use an “ordinary” user account most of the time
  - Except when you are making changes to your system

Controlled escalation of privilege

- `sudo`
  - Allows certain users to run certain things as other users
    - Why? Because the users normally cannot run those things
    - That's why I say "escalation of privilege"
  - Everything is logged
  - See `http://xkcd.com/838/`
  - Be careful with this

- setuid bit programs
  - Allows anyone to run this program as the file owner
    - Again — necessary to allow users to do things they normally could not
  - Be careful with this

Uncontrolled escalation of privilege

This is usually very very bad

- Suppose a cracker wants to execute some commands as another user
    - “Boy, I really want to read that file that I am not allowed to”
    - Or worse. . .

Uncontrolled escalation of privilege

This is usually very very bad

- Suppose a cracker wants to execute some commands as another user
    - “Boy, I really want to read that file that I am not allowed to”
    - Or worse. . .
- How this might be done:
    1. Find a command that can be run as that user or `root`
        - Using `sudo` or a setuid program
    2. Within that command, try to execute other commands
        - We will discuss ways to do this
    3. The commands will run as the other user

Uncontrolled escalation of privilege

This is usually very very bad

- Suppose a cracker wants to execute some commands as another user
  - "Boy, I really want to read that file that I am not allowed to"
  - Or worse. . .
- How this might be done:
  1. Find a command that can be run as that user or `root`
     - Using `sudo` or a setuid program
  2. Within that command, try to execute other commands
     - We will discuss ways to do this
  3. The commands will run as the other user
- setuid `root` programs are the natural target
  - Most users do not have `sudo` privileges
  - Everyone can execute setuid `root` programs
  - `root` has sufficient privileges for anything

Executing commands within another command

- Super easy way: the command has “drop to shell” ability
  - These should never be setuid `root` or allowed in `sudo`

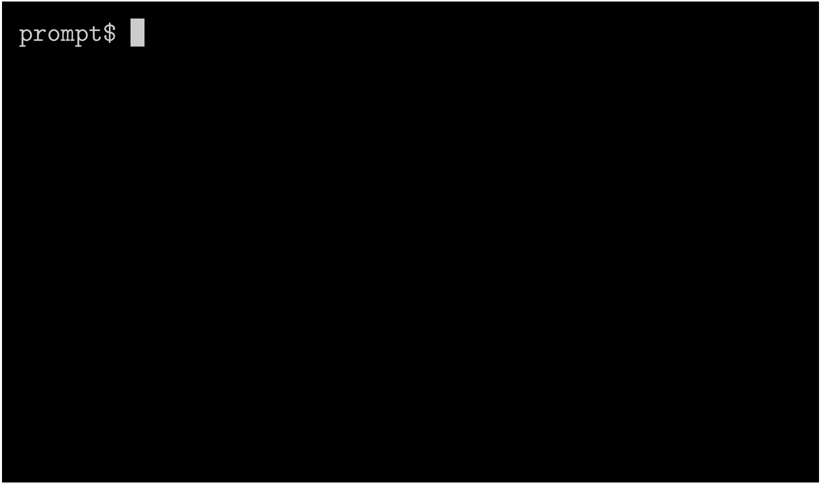
Executing commands within another command

- Super easy way: the command has "drop to shell" ability
  - These should never be setuid `root` or allowed in `sudo`
- Almost as easy: the command is actually a shell script
  - It is not easy to write a secure shell script
  - Can be cracked if any utility used in the script is not specified with an absolute pathname:
    1. Have a look at the shell script
    2. Choose some utility used in the script
    3. Write a script with the same name
    4. Change path
    5. Run the setuid / sudo script
  - Many systems ignore the setuid bit for shell scripts for this very reason

Executing commands within another command

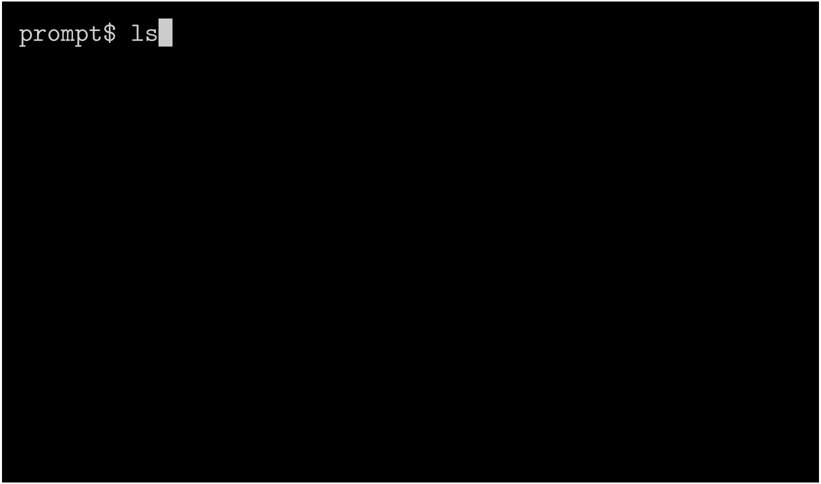
- Super easy way: the command has “drop to shell” ability
  - These should never be setuid `root` or allowed in `sudo`
- Almost as easy: the command is actually a shell script
  - It is not easy to write a secure shell script
  - Can be cracked if any utility used in the script is not specified with an absolute pathname:
    1. Have a look at the shell script
    2. Choose some utility used in the script
    3. Write a script with the same name
    4. Change path
    5. Run the setuid / sudo script
  - Many systems ignore the setuid bit for shell scripts for this very reason
- Harder: exploit security holes in the program
  - We will see an example of this in a minute

Example: Injecting commands into a shell script



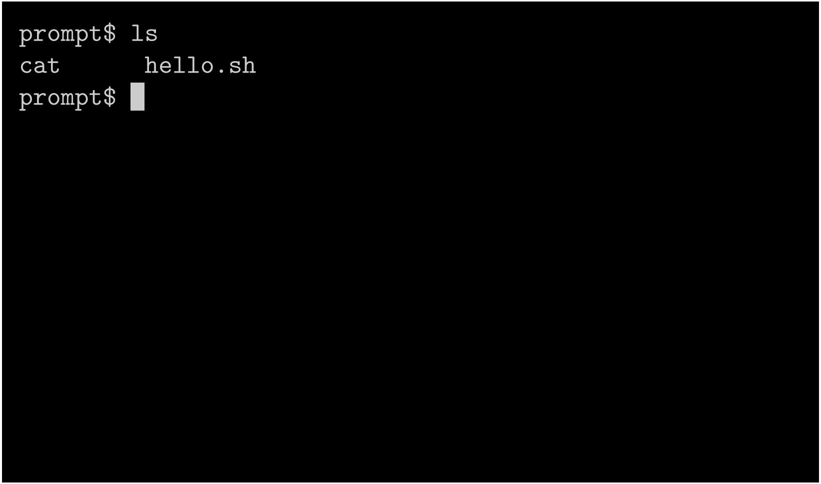
```
prompt$ █
```

Example: Injecting commands into a shell script



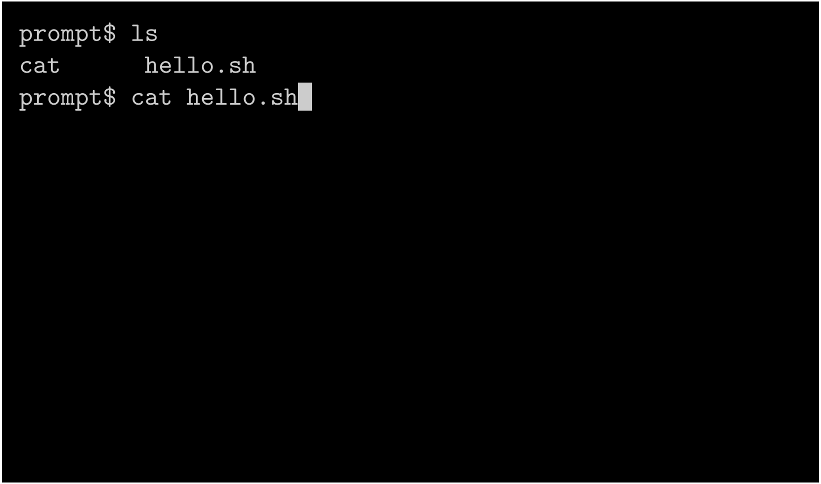
```
prompt$ ls
```

Example: Injecting commands into a shell script



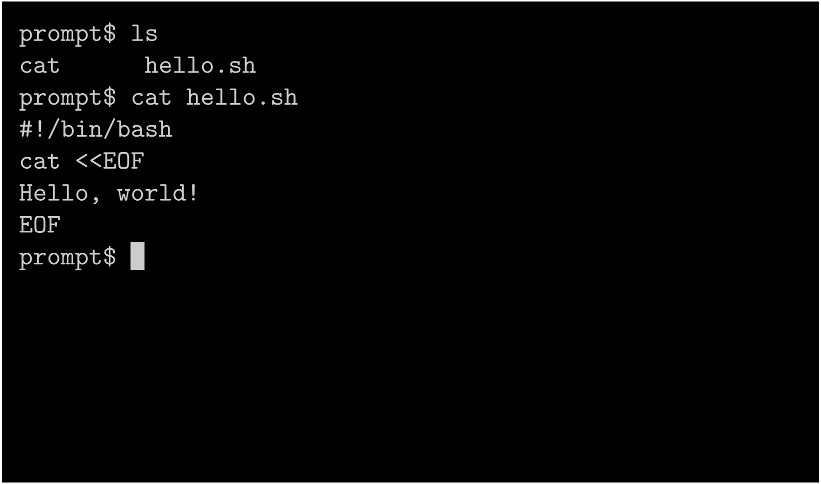
```
prompt$ ls
cat       hello.sh
prompt$ 
```

Example: Injecting commands into a shell script



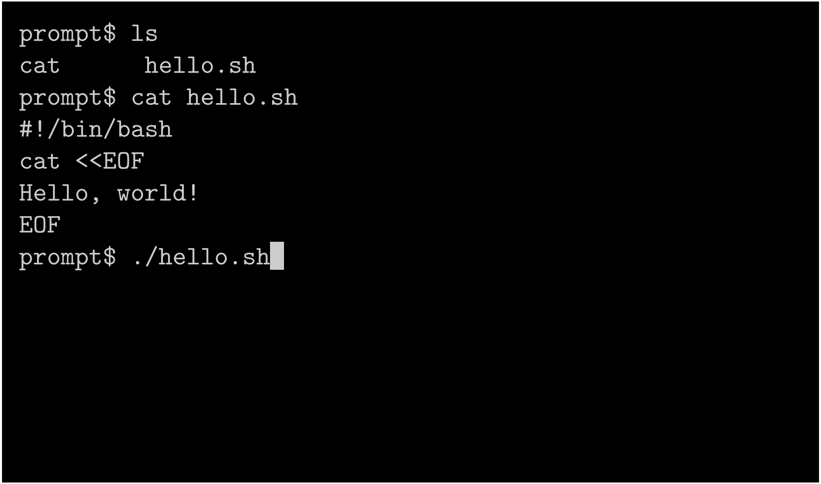
```
prompt$ ls
cat      hello.sh
prompt$ cat hello.sh
```

Example: Injecting commands into a shell script



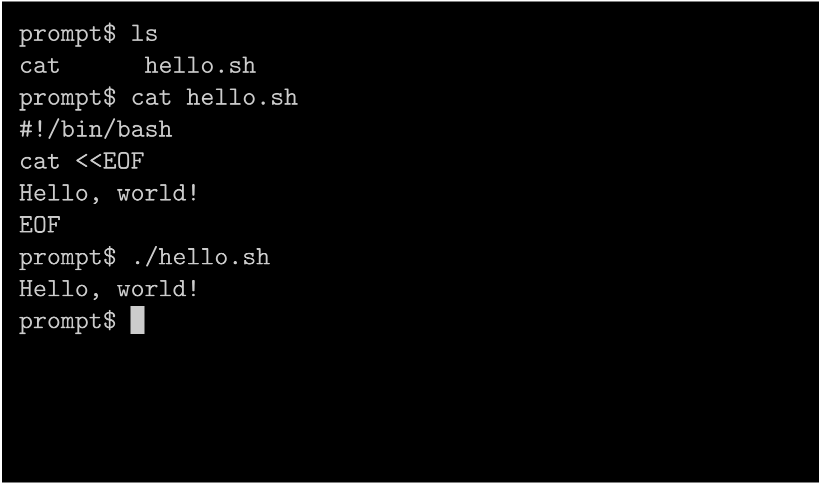
```
prompt$ ls
cat      hello.sh
prompt$ cat hello.sh
#!/bin/bash
cat <<EOF
Hello, world!
EOF
prompt$ 
```

Example: Injecting commands into a shell script



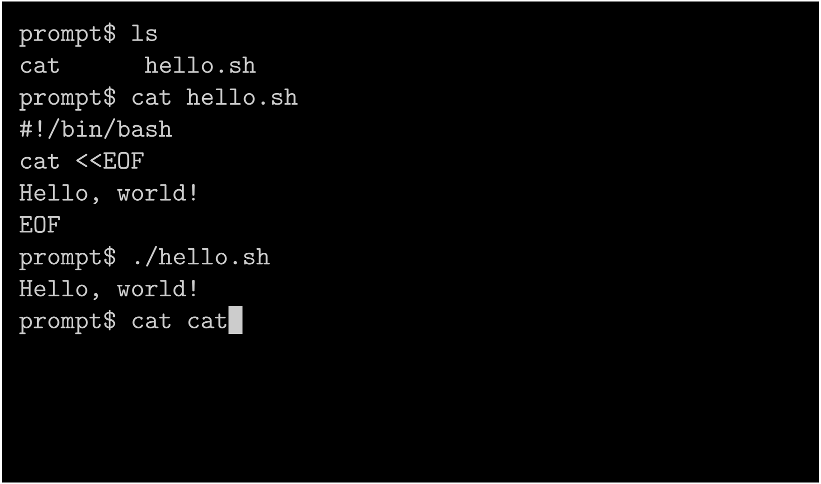
```
prompt$ ls
cat       hello.sh
prompt$ cat hello.sh
#!/bin/bash
cat <<EOF
Hello, world!
EOF
prompt$ ./hello.sh
```

Example: Injecting commands into a shell script



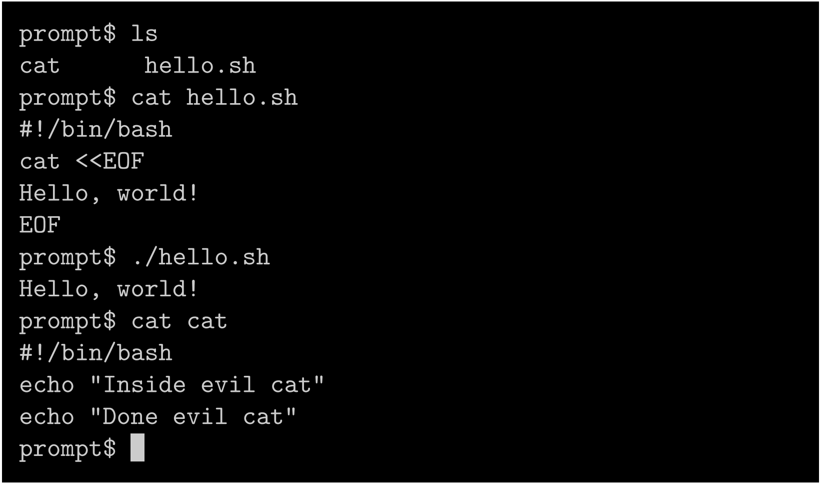
```
prompt$ ls
cat      hello.sh
prompt$ cat hello.sh
#!/bin/bash
cat <<EOF
Hello, world!
EOF
prompt$ ./hello.sh
Hello, world!
prompt$ 
```

Example: Injecting commands into a shell script



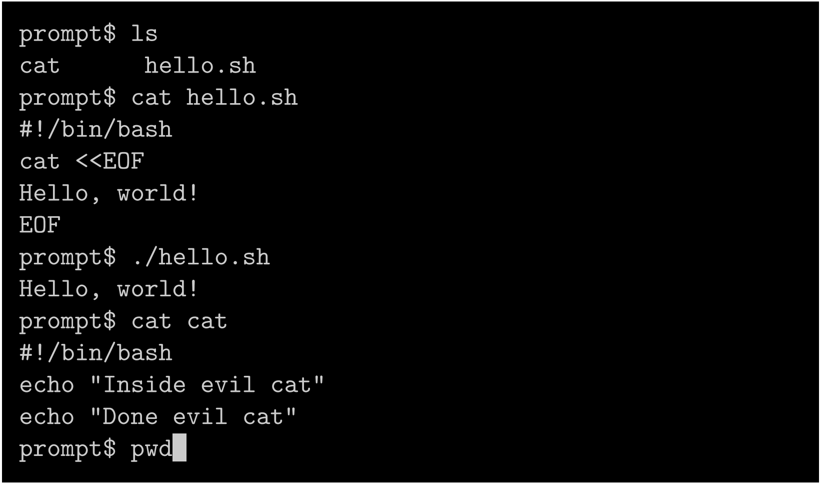
```
prompt$ ls
cat      hello.sh
prompt$ cat hello.sh
#!/bin/bash
cat <<EOF
Hello, world!
EOF
prompt$ ./hello.sh
Hello, world!
prompt$ cat cat
```

Example: Injecting commands into a shell script



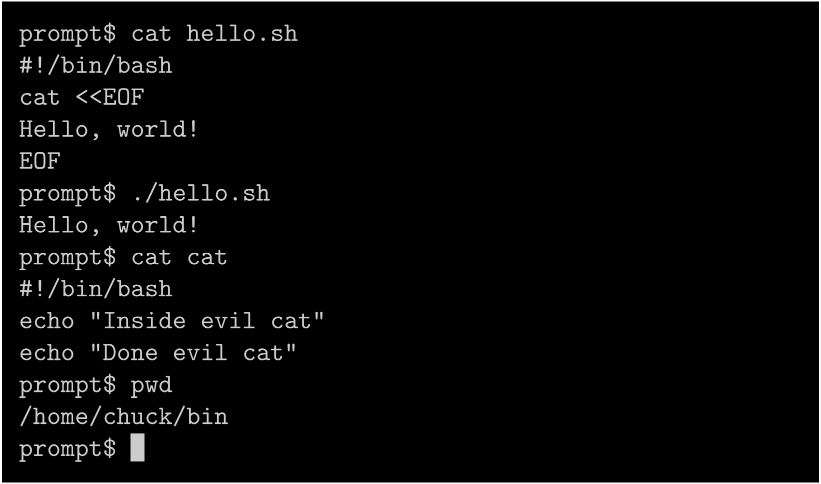
```
prompt$ ls
cat      hello.sh
prompt$ cat hello.sh
#!/bin/bash
cat <<EOF
Hello, world!
EOF
prompt$ ./hello.sh
Hello, world!
prompt$ cat cat
#!/bin/bash
echo "Inside evil cat"
echo "Done evil cat"
prompt$ █
```

Example: Injecting commands into a shell script



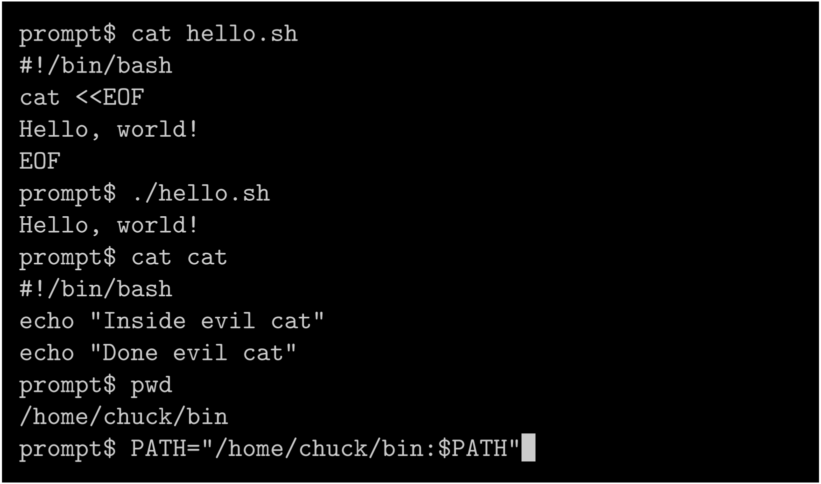
```
prompt$ ls
cat      hello.sh
prompt$ cat hello.sh
#!/bin/bash
cat <<EOF
Hello, world!
EOF
prompt$ ./hello.sh
Hello, world!
prompt$ cat cat
#!/bin/bash
echo "Inside evil cat"
echo "Done evil cat"
prompt$ pwd
```

Example: Injecting commands into a shell script



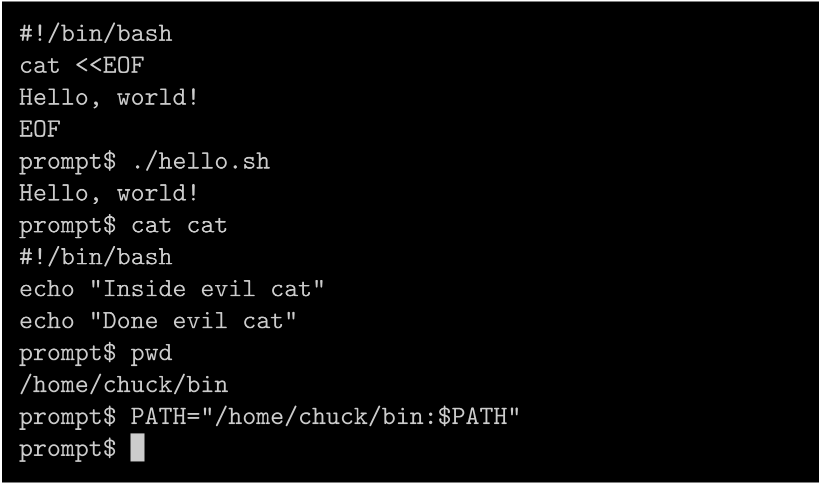
```
prompt$ cat hello.sh
#!/bin/bash
cat <<EOF
Hello, world!
EOF
prompt$ ./hello.sh
Hello, world!
prompt$ cat cat
#!/bin/bash
echo "Inside evil cat"
echo "Done evil cat"
prompt$ pwd
/home/chuck/bin
prompt$ █
```

Example: Injecting commands into a shell script



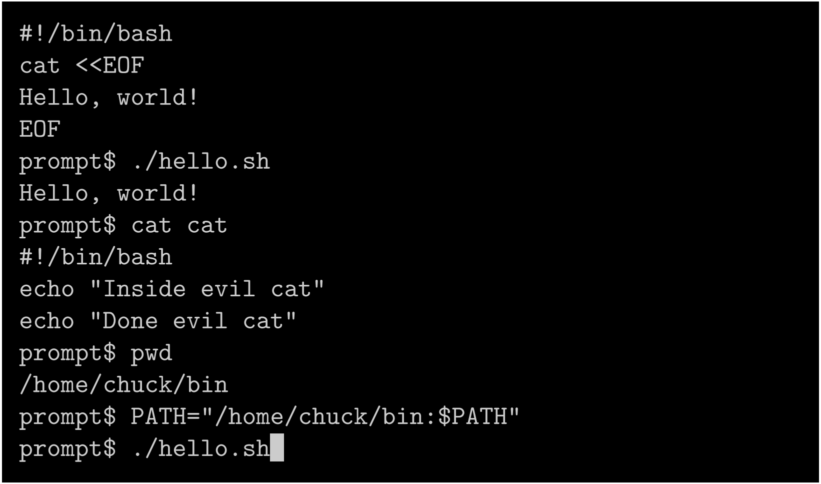
```
prompt$ cat hello.sh
#!/bin/bash
cat <<EOF
Hello, world!
EOF
prompt$ ./hello.sh
Hello, world!
prompt$ cat cat
#!/bin/bash
echo "Inside evil cat"
echo "Done evil cat"
prompt$ pwd
/home/chuck/bin
prompt$ PATH="/home/chuck/bin:$PATH"
```

Example: Injecting commands into a shell script



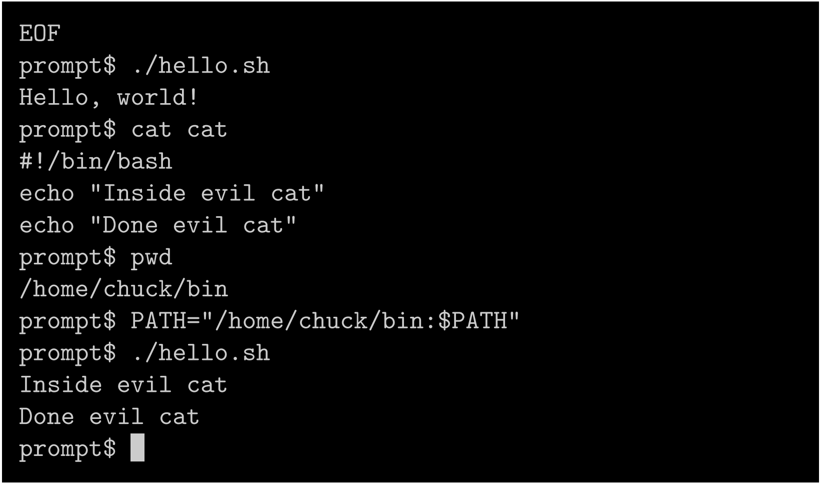
```
#!/bin/bash
cat <<EOF
Hello, world!
EOF
prompt$ ./hello.sh
Hello, world!
prompt$ cat cat
#!/bin/bash
echo "Inside evil cat"
echo "Done evil cat"
prompt$ pwd
/home/chuck/bin
prompt$ PATH="/home/chuck/bin:$PATH"
prompt$ █
```

Example: Injecting commands into a shell script



```
#!/bin/bash
cat <<EOF
Hello, world!
EOF
prompt$ ./hello.sh
Hello, world!
prompt$ cat cat
#!/bin/bash
echo "Inside evil cat"
echo "Done evil cat"
prompt$ pwd
/home/chuck/bin
prompt$ PATH="/home/chuck/bin:$PATH"
prompt$ ./hello.sh
```

Example: Injecting commands into a shell script



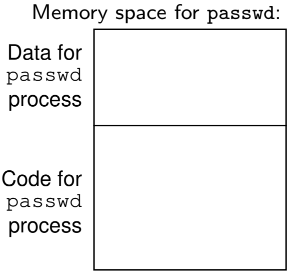
```
EOF
prompt$ ./hello.sh
Hello, world!
prompt$ cat cat
#!/bin/bash
echo "Inside evil cat"
echo "Done evil cat"
prompt$ pwd
/home/chuck/bin
prompt$ PATH="/home/chuck/bin:$PATH"
prompt$ ./hello.sh
Inside evil cat
Done evil cat
prompt$
```

The buffer overflow exploit

- A classic exploit based on a security hole
- How it works (greatly simplified version):
  - The memory space of a program includes
    - Data (program variables)
    - Code (machine language instructions)
  - The program copies a user–entered string into a buffer
  - The programmer was lazy, and did not first check that the string would fit
  - The cracker enters a diabolical string. . .

The buffer overflow exploit, illustrated

Again, greatly simplified

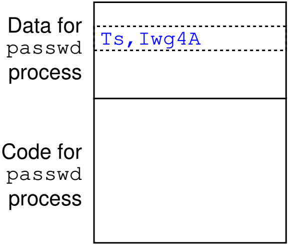


- Suppose the `passwd` utility has this security hole

The buffer overflow exploit, illustrated

Again, greatly simplified

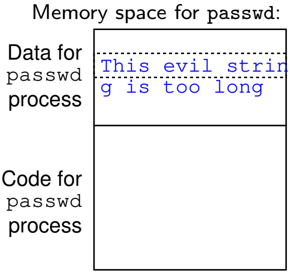
Memory space for passwd:



- Suppose the `passwd` utility has this security hole
- Normally: user string fits in the buffer

The buffer overflow exploit, illustrated

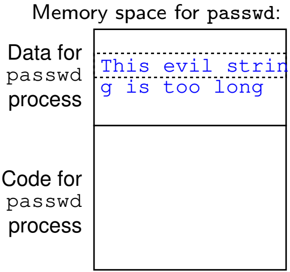
Again, greatly simplified



- Suppose the passwd utility has this security hole
- Normally: user string fits in the buffer
- Long strings will overwrite other data

The buffer overflow exploit, illustrated

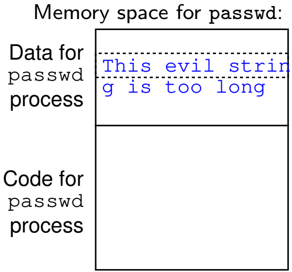
Again, greatly simplified



- Suppose the passwd utility has this security hole
- Normally: user string fits in the buffer
- Long strings will overwrite other data
  - “But Java will throw an exception. . .”

The buffer overflow exploit, illustrated

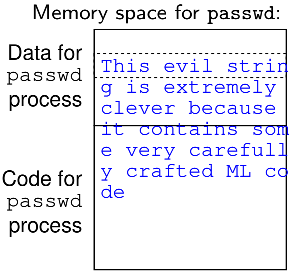
Again, greatly simplified



- Suppose the `passwd` utility has this security hole
- Normally: user string fits in the buffer
- Long strings will overwrite other data
  - "But Java will throw an exception…"
  - In C you have to do everything yourself!

The buffer overflow exploit, illustrated

Again, greatly simplified



Memory space for `passwd`:

Data for `passwd` process

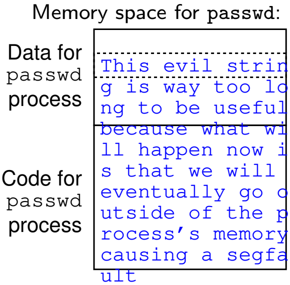
Code for `passwd` process

```
This evil strin
g is extremely
clever because
it contains som
e very carefull
y crafted ML co
de
```

- Suppose the `passwd` utility has this security hole
- Normally: user string fits in the buffer
- Long strings will overwrite other data
  - "But Java will throw an exception. . ."
  - In C you have to do everything yourself!
- Longer strings will overwrite code
  - I can put binary code in my string and get the process to execute my code (which will be: open a terminal window)

The buffer overflow exploit, illustrated

Again, greatly simplified



Memory space for `passwd`:

Data for `passwd` process

Code for `passwd` process

```
This evil strin
g is way too lo
ng to be useful
because what wi
ll happen now i
s that we will 
eventually go o
utside of the p
rocess's memory
causing a segfa
ult
```

- Suppose the `passwd` utility has this security hole
- Normally: user string fits in the buffer
- Long strings will overwrite other data
  - "But Java will throw an exception…"
  - In C you have to do everything yourself!
- Longer strings will overwrite code
  - I can put binary code in my string and get the process to execute my code (which will be: open a terminal window)
- Really long strings: will cause a segfault

Is the buffer overflow attack feasible?

Skeptics: “There is no way you could craft that magic string. The length has to be *just right* and the code you want to execute has to be *perfect machine language* and in the right spot.”

Is the buffer overflow attack feasible?

Skeptics: “There is no way you could craft that magic string. The length has to be *just right* and the code you want to execute has to be *perfect machine language* and in the right spot.”

1. I don’t need to know machine language, that’s what compilers are for
2. I don’t need to get it right on the first try
3. I can write a script to try lots of strings until I get it right
    - Segmentation fault: my string was too long

Is the buffer overflow attack feasible?

Skeptics: “There is no way you could craft that magic string. The length has to be *just right* and the code you want to execute has to be *perfect machine language* and in the right spot.”

1. I don’t need to know machine language, that’s what compilers are for
2. I don’t need to get it right on the first try
3. I can write a script to try lots of strings until I get it right
   - Segmentation fault: my string was too long

Yes, buffer overflows are feasible

Preventing buffer overflow

- Don't write buggy code
- Keep your system software up to date
  - Bug fixes may also patch security holes
- Modern compilers and/or kernels may protect against this
  - E.g., better memory protection so that code space cannot be overwritten
- Modern compilers may complain when code uses the offending library functions (e.g., `strcpy`)
- SELinux may prevent this
  - More on SELinux later...

Should I care about this type of exploit?

- Suppose there is only one user account: mine
- I also have `root` access because it is my machine
- There is no point for me to try any of these exploits
- Do I need to worry about this?

Should I care about this type of exploit?

- Suppose there is only one user account: mine
- I also have `root` access because it is my machine
- There is no point for me to try any of these exploits
- Do I need to worry about this?

- Yes — if I connect this machine to a network
- Suppose there is a hole in Apache and an intruder can obtain a shell running as user `apache`
- Step 2 will be to escalate privileges

What is Network Security?

What is Network Security?

- Keeping the network services secure from unauthorized access
- Preventing exploits through security holes in network services
- Applies to any service that communicates via IP

An example exploit (an old one)

`http://technet.microsoft.com/en-us/security/bulletin/ms06-036`

- Buffer overflow possibility on DHCP clients in Microsoft OSes
- Discovered in 2006 — your system should be patched by now!
- How this exploit works:
  1. Cracker sets up a rogue DHCP server
  2. Rogue DHCP server runs DHCP protocol except: sends a carefully crafted packet back
  3. The carefully crafted packet causes a buffer overflow on the client, allowing code execution

An example exploit (an old one)

`http://technet.microsoft.com/en-us/security/bulletin/ms06-036`

- Buffer overflow possibility on DHCP clients in Microsoft OSes
- Discovered in 2006 — your system should be patched by now!
- How this exploit works:
  1. Cracker sets up a rogue DHCP server
  2. Rogue DHCP server runs DHCP protocol except:
     sends a carefully crafted packet back
  3. The carefully crafted packet causes a buffer overflow on the client, allowing code execution
- How feasible is this?
  - It is easy to set up a rogue DHCP server
    if I have physical access to your network
    - Client and server(s) broadcast over common ethernet segment
  - For a wired home network — probably not a concern
  - But do you use wireless anywhere?

Other general types of attacks

Man in the middle

Based on the following:

- Cracker controls a router machine
- Cracker can intercept packets between a client and server
- Cracker can alter the packets

Other general types of attacks

Man in the middle

Based on the following:

- Cracker controls a router machine
- Cracker can intercept packets between a client and server
- Cracker can alter the packets

Denial of service (DoS)

- Goal: reduce a server's ability to process *legitimate* requests
- Common approach: flood the server with requests
- Distributed DoS: flood the server from different clients

Why are they after me again?

In denial: “The Internet is huge, crackers are not going to target my unknown machine, you are being paranoid.”

Why are they after me again?

In denial: “The Internet is huge, crackers are not going to target my unknown machine, you are being paranoid.”

- “Just because I’m paranoid doesn’t mean they’re not out to get me”

Why are they after me again?

In denial: “The Internet is huge, crackers are not going to target my unknown machine, you are being paranoid.”

- “Just because I’m paranoid doesn’t mean they’re not out to get me”
- They are not after you personally (unless you are Google, Microsoft, Wells Fargo, . . .)
- They are after unsecured machines in general
  - And they will find yours

Why are they after me again?

In denial: "The Internet is huge, crackers are not going to target my unknown machine, you are being paranoid."

- "Just because I'm paranoid doesn't mean they're not out to get me"
- They are not after you personally
  (unless you are Google, Microsoft, Wells Fargo, . . .)
- They are after unsecured machines in general
  - And they will find yours

Why?

Why are they after me again?

In denial: "The Internet is huge, crackers are not going to target my unknown machine, you are being paranoid."

- "Just because I'm paranoid doesn't mean they're not out to get me"
- They are not after you personally
  (unless you are Google, Microsoft, Wells Fargo, . . .)
- They are after unsecured machines in general
  - And they will find yours

Why?

- Maybe after your personal data
- Maybe to use your machine to attack a larger target

Why are they after me again?

In denial: "The Internet is huge, crackers are not going to target my unknown machine, you are being paranoid."

- "Just because I'm paranoid doesn't mean they're not out to get me"
- They are not after you personally
  (unless you are Google, Microsoft, Wells Fargo, . . .)
- They are after unsecured machines in general
  - And they will find yours

Why?

- Maybe after your personal data
- Maybe to use your machine to attack a larger target
- Another compelling reason — botnets

Botnets

- A botnet is a collection of compromised machines (bots)
- Each bot has an Internet connection and runs malware
- The botnet is controlled remotely by a “bot herder”
- Botnets are typically used for
  - Distributed denial of service attacks
  - Bulk spam
  - Adware — replaces web page ads in browsers on the bots
  - Recruiting more bots
- Botnets are often idle to avoid detection
- A botnet containing 10,000 bots is a small one

Botnets

- A botnet is a collection of compromised machines (bots)
- Each bot has an Internet connection and runs malware
- The botnet is controlled remotely by a “bot herder”
- Botnets are typically used for
  - Distributed denial of service attacks
  - Bulk spam
  - Adware — replaces web page ads in browsers on the bots
  - Recruiting more bots
- Botnets are often idle to avoid detection
- A botnet containing 10,000 bots is a small one
- Largest discovered botnet had about 30 million bots (according to Wikipedia, anyway)

General principles for network security

- Only run services that you need
  - Any running network service is a potential entry point
- Keep those services up to date
  - `yum upgrade` is your friend
- Extend "principle of least privilege" to network services
  - Allow services only for appropriate IP addresses
  - Use secure TCP wrappers ("`ALL:ALL`" in `hosts.deny`)
  - Use a tight firewall

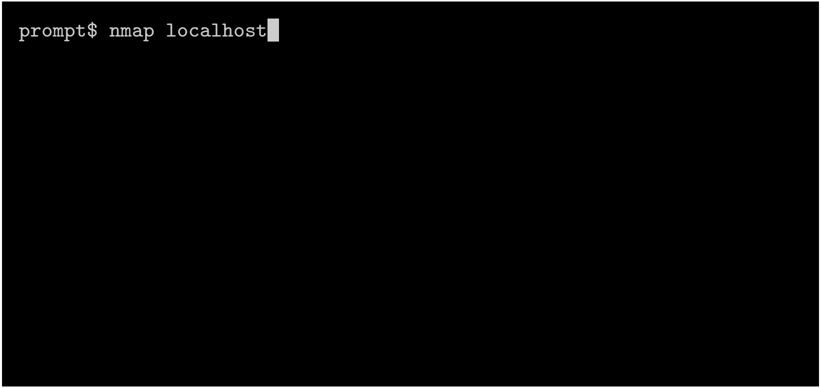
nmap utility

- Port scanner — shows what ports are open on a host
- Useful tool when securing a machine
- Also useful for crackers to see potential entry points
  - So some systems do not take kindly to being scanned

```
prompt$ 
```

nmap utility

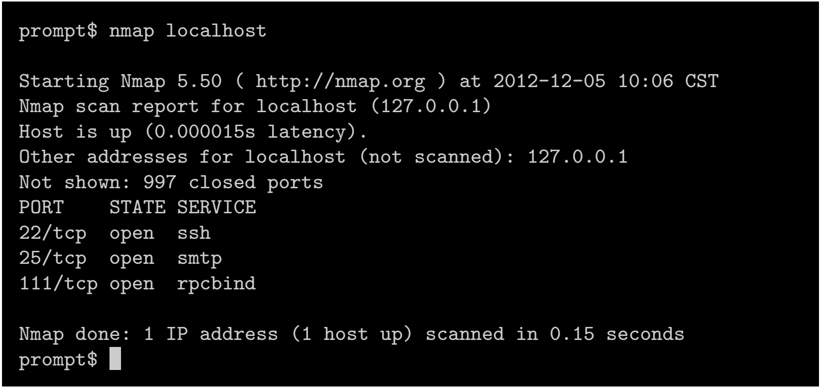
- Port scanner — shows what ports are open on a host
- Useful tool when securing a machine
- Also useful for crackers to see potential entry points
  - So some systems do not take kindly to being scanned



```
prompt$ nmap localhost
```

nmap utility

- Port scanner — shows what ports are open on a host
- Useful tool when securing a machine
- Also useful for crackers to see potential entry points
  - So some systems do not take kindly to being scanned



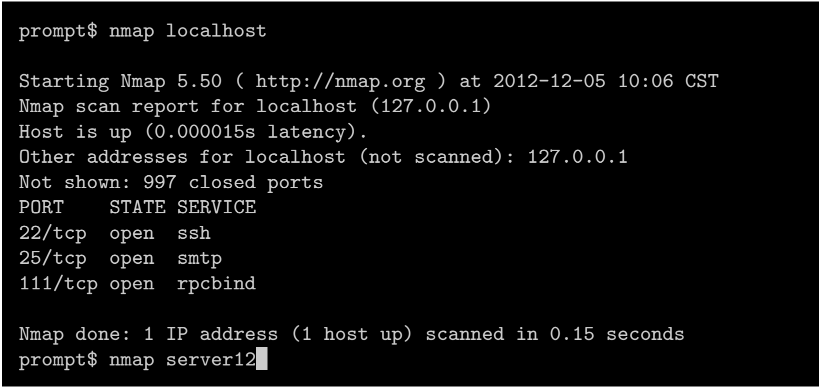
```
prompt$ nmap localhost

Starting Nmap 5.50 ( http://nmap.org ) at 2012-12-05 10:06 CST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000015s latency).
Other addresses for localhost (not scanned): 127.0.0.1
Not shown: 997 closed ports
PORT    STATE SERVICE
22/tcp  open  ssh
25/tcp  open  smtp
111/tcp open  rpcbind

Nmap done: 1 IP address (1 host up) scanned in 0.15 seconds
prompt$ 
```

nmap utility

- Port scanner — shows what ports are open on a host
- Useful tool when securing a machine
- Also useful for crackers to see potential entry points
  - So some systems do not take kindly to being scanned



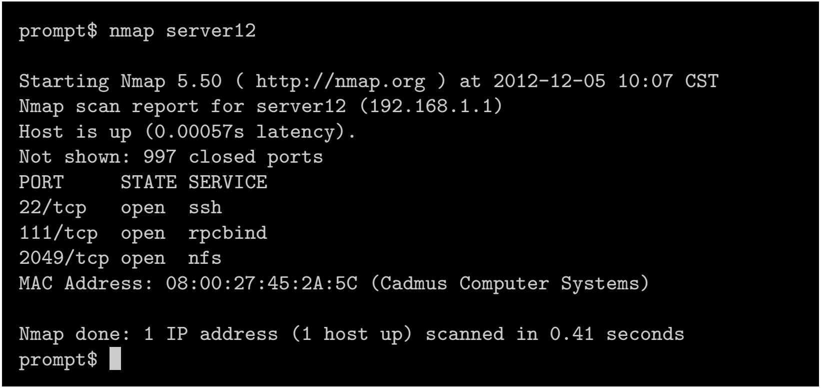
```
prompt$ nmap localhost

Starting Nmap 5.50 ( http://nmap.org ) at 2012-12-05 10:06 CST
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000015s latency).
Other addresses for localhost (not scanned): 127.0.0.1
Not shown: 997 closed ports
PORT    STATE SERVICE
22/tcp  open  ssh
25/tcp  open  smtp
111/tcp open  rpcbind

Nmap done: 1 IP address (1 host up) scanned in 0.15 seconds
prompt$ nmap server12
```

nmap utility

- Port scanner — shows what ports are open on a host
- Useful tool when securing a machine
- Also useful for crackers to see potential entry points
    - So some systems do not take kindly to being scanned



```
prompt$ nmap server12

Starting Nmap 5.50 ( http://nmap.org ) at 2012-12-05 10:07 CST
Nmap scan report for server12 (192.168.1.1)
Host is up (0.00057s latency).
Not shown: 997 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
111/tcp  open  rpcbind
2049/tcp open  nfs
MAC Address: 08:00:27:45:2A:5C (Cadmus Computer Systems)

Nmap done: 1 IP address (1 host up) scanned in 0.41 seconds
prompt$
```

SELinux

- Security Enhanced Linux
- Developed by the NSA (National Security Agency)
- Goal: add kernel support for mandatory access controls
- Uses contexts, consisting of
  1. A role
  2. A user (not necessarily a userID, can be a service)
  3. A domain or type
- Policy rules: explicit permission to perform an action
  - What domains a user must possess to perform an action
  - Actions include read, write, execute, and others
- Essentially: “like an internal firewall”
- Has been included with Fedora since Fedora Core 2

SELinux (2)

- SELinux has 3 modes (set in `/etc/sysconfig/selinux`)

  Disabled : SELinux policy is not enforced

  Permissive : like "disabled" but prints warnings

  - Useful for debugging your policy rules

  Enforcing : SELinux policy is enforced

  - The default mode with Fedora

SELinux (2)

- SELinux has 3 modes (set in `/etc/sysconfig/selinux`)

  Disabled : SELinux policy is not enforced

  Permissive : like “disabled” but prints warnings
  - Useful for debugging your policy rules

  Enforcing : SELinux policy is enforced
  - The default mode with Fedora
- For many VMs this semester: I set the mode to “disabled”
  - Most of the networking VMs
  - The assignment with a “new drive”
  - In short: anything where the assignment would not work

Intrusion detection

Suppose you *suspect* that someone has compromised your machine (obtained `root` access). How do you check this?

Intrusion detection

Suppose you *suspect* that someone has compromised your machine (obtained `root` access). How do you check this?

- Look for new accounts in `/etc/passwd`
  - Maybe `cat /etc/passwd` and inspect
  - Or maybe `grep -v nologin /etc/passwd`
- Look for new files
  - Maybe `ls /home`
- Look for unknown running processes
  - Maybe `ps aux | grep -v root`
- Look in system logs (e.g., `/var/log/secure`)

Intrusion detection

Suppose you *suspect* that someone has compromised your machine (obtained `root` access). How do you check this?

- Look for new accounts in `/etc/passwd`
  - Maybe `cat /etc/passwd` and inspect
  - Or maybe `grep -v nologin /etc/passwd`
- Look for new files
  - Maybe `ls /home`
- Look for unknown running processes
  - Maybe `ps aux | grep -v root`
- Look in system logs (e.g., `/var/log/secure`)

But. . .

1. How do you know where to look?

Intrusion detection

Suppose you *suspect* that someone has compromised your machine (obtained `root` access). How do you check this?

- Look for new accounts in `/etc/passwd`
  - Maybe `cat /etc/passwd` and inspect
  - Or maybe `grep -v nologin /etc/passwd`
- Look for new files
  - Maybe `ls /home`
- Look for unknown running processes
  - Maybe `ps aux | grep -v root`
- Look in system logs (e.g., `/var/log/secure`)

But. . .

1. How do you know where to look?
2. Why should system logs still be intact?

Intrusion detection

Suppose you *suspect* that someone has compromised your machine (obtained `root` access). How do you check this?

- Look for new accounts in `/etc/passwd`
  - Maybe `cat /etc/passwd` and inspect
  - Or maybe `grep -v nologin /etc/passwd`
- Look for new files
  - Maybe `ls /home`
- Look for unknown running processes
  - Maybe `ps aux | grep -v root`
- Look in system logs (e.g., `/var/log/secure`)

But…

1. How do you know where to look?
2. Why should system logs still be intact?
3. Why should `cat`, `grep`, `ls`, `ps` still work?

Rootkits

What is a "rootkit"?

- Tools to allow a cracker to
  1. Obtain root access over the system
  2. Conceal these activities from the real sysadmin
- May be installed using a script or by hand
- Could include "new and improved" versions of
  - `cat` and `grep`
  - `ls` and `ps`
  - `su` and `sudo`
  - `login` and `passwd`
  - `gcc`
  - the kernel and kernel modules
- Good ones will also cover tracks in log files

Detecting a rootkit — properly

Suppose you *suspect* that someone has compromised your machine (obtained `root` access). How do you check this?

Detecting a rootkit — properly

Suppose you *suspect* that someone has compromised your machine (obtained `root` access). How do you check this?

- Boot using a live CD then search for the rootkit
  - Live CD gives the "proper" versions of utilities
  - The rootkit cannot hide itself

Detecting a rootkit — properly

Suppose you *suspect* that someone has compromised your machine (obtained `root` access). How do you check this?

- Boot using a live CD then search for the rootkit
  - Live CD gives the "proper" versions of utilities
  - The rootkit cannot hide itself
- `chkrootkit`: utility to search for known rootkits
  - `http://freecode.com/projects/chkrootkit`
  - Again, run this from a live CD

File integrity software: idea

- Can be used to check for modified files
- Produces a checksum value for a configured set of files
- The checksum is saved for “pristine” files
- Periodically recompute the checksum and compare
- If the checksums differ — something has changed
- Stored checksum must not be writable by crackers
  - Tradeoff between security and convenience
    - This is common
  - E.g.: burn checksum to CDR and mount
    - Must re-burn whenever changes are made

File integrity software: examples

Tripwire

- Open source
- `http://sourceforge.net/projects/tripwire/`
- Encrypts database with a passphrase
- Database is readable without the passphrase
- Database updates require the passphrase

Integrit

- Also open source
- `http://sourceforge.net/projects/integrit/`
- Does not appear to be as active a project as Tripwire

Recovery

Two choices

1. Attempt to remove the rootkit
   - Tricky — must get all corrupted utilities out
2. Do a clean reinstall
   - I would strongly recommend doing this
   - You back up your data, right?
   - Might be able to keep your `/home` partition. . .

Simple steps to make crackers' work more difficult

- Use a different, memorized, strong password for each system
- Avoid logging in as `root` or administrator
  - Use `su` or `sudo` instead
- Remember the principle of least privilege
- Minimize the amount of software installed
- Minimize the number of running services
- Keep system software up to date
- Use security–enhanced tools whenever possible
  - SELinux, IPTables, TCP wrappers
- Encrypt network traffic whenever possible

One last truth in security

One last truth in security

Systems are only as secure as the weakest component

One last truth in security

Systems are only as secure as the weakest component

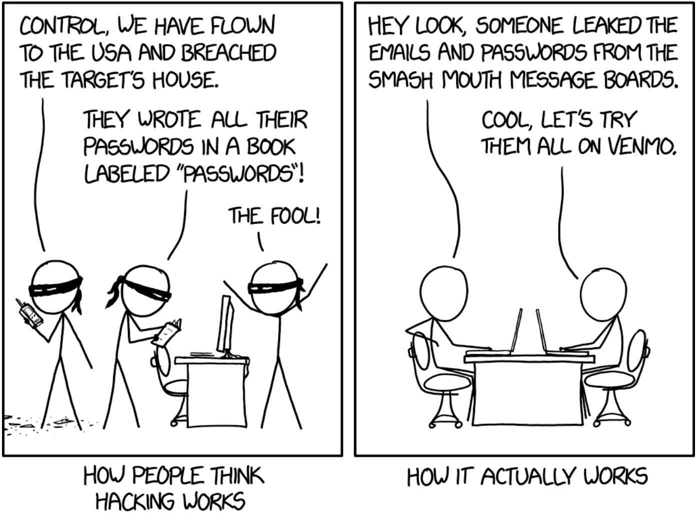
- Weakest component tends to be:

One last truth in security

Systems are only as secure as the weakest component

- Weakest component tends to be: users

An appropriate xkcd comic: http://www.xkcd.com/2176


CONTROL, WE HAVE FLOWN TO THE USA AND BREACHED THE TARGET'S HOUSE.
THEY WROTE ALL THEIR PASSWORDS IN A BOOK LABELED "PASSWORDS"!
THE FOOL!
HOW PEOPLE THINK HACKING WORKS
HEY LOOK, SOMEONE LEAKED THE EMAILS AND PASSWORDS FROM THE SMASH MOUTH MESSAGE BOARDS.
COOL, LET'S TRY THEM ALL ON VENMO.
HOW IT ACTUALLY WORKS

End of lecture