Project – netGoat

Learning Objectives

After completing these activities you should be able to:

- Design, develop, debug, and document systems level programs in C using structured programming techniques.
- Develop a program to execute in a UNIX environment.
- Develop programs that utilize inter process communication.

Description

WARNING: *As a student in this class, you will learn concepts and gain experience with tools that could be used unethically. DO NOT use knowledge or experience gained for unethical purposes. You MAY NOT use tools and techniques learned in this class to violate USNA policy, or any other government restrictions on information system use. You should never employ offensive cyber operations on any information system without the express written consent of the information system owner or legal authority.*

In this project you will design a program that could be used in various cyber operations, especially when combined with other tools. Through this project you will develop an enhanced version of netcat (`nc(1)`).

You will develop your program incrementally, adding features at various points through the course as we cover topics in the course. Not every topic discussed in the course material will be directly used in the project. Most of the project milestones will draw on material covered in programming lab assignments. At various points you will have options regarding what you decide to implement in your `netGoat` program. You will explain your decisions in both a written report and oral presentation by describing how your `netGoat` could be used in cyber operations.

Assignment Information

Programming Project: netGoat

Assignment Type:	Programming - Project	Collaboration Policy:	Default
Assignment		netGoat	
Due			
Section		Course Server Time / Date	
All		Thursday, 1200 04 May 2018	

This project is a small group project (2-3 members, see Group Assignments).

You will be given some time in class to work on the project, but ,like labs, it is expected that you will need to use outside class time to work on and complete the project deliverables. During in-class project time it is expected that all group members will be actively working on the project. All team members we be expected to speak to all components of the project regardless of the division of labor decided on between the team members. Individual team member knowledge of your project is part of your (team collective) grade.

Deliverables

Explicit Due Dates

- Group Name:

  Choose a team name for your group (creativity encouraged). Email your group name to your instructor.
  By: 1600 01 Mar 2018
  Note: These are information systems connected to a government network, choose a team name accordingly.

- Repository Location:

  Email your instructor the absolute path to the project directory on `mope` (see Milestone 0).
  By: 1600 01 Mar 2018

- Oral Presentation:

  In-class presentation schedule to be determined.
  By: 02 May 2018

- Source Code:

  Committed and pushed to team's master repository.
  By: 1200 04 May 2018

- Written Report:

  Printed copy to your instructor.
  By: 1200 04 May 2018

- Peer Review:

  Printed copy to your instructor.
  By: 1200 04 May 2018

Project Grade Weighting

- Milestone 0: 0%
- Milestone 1: 5%
- Milestone 2: 10%
- Milestone 3: 10%
- Milestone 4: 10%
- Milestone 5: 10%
- Milestone 6: 10%
- Milestone 7: 10%
- Milestone 8: 5%
- Oral Presentation: 10%
- Written Report: 10%
- Peer Evaluation: 10%

Milestones

[–] Milestone: 0 – Project Setup

Before you begin working on a collaborative project you need to steup a work environment that supports collaboration. The following steps should be taken in order to setup your team's project work environment. For collaborative coding projects two things are needed: a project workspace (storage space within a file system) that is accessible by all team members, and a Revision Control System (RCS) to manage project changes from various team members. Typically a project workspace that is under revision control is called a *repository*. The commonly used RCSes work outside of the editors and compilation process that you are used to (so you don't have to change the way in which you write and test code), but there are plug-ins and extensions for common RCSes, and development tools that can be used to streamline the development and project management process.

For this project we will use Git (`git`), Git is an open source, free, Distributed Revision Control System (DRCS). The *distributed* refers to the way in which the repository is managed, each *clone* of a repository in a DRCS is a complete copy of the project. This allows for quicker completion of local tasks; in exchange for using more local storage space. Git is installed on all department Linux lab machines, and is available for various flavors of UNIX, Windows, and Mac OS X.
Note: We will discuss what some of these commands are doing through future course work.

note: Course Material & Revision Control The material for this course is maintained using Git. The course repository contains material for the course website, and material that is not available through the course website. The Linux kernel is maintained using Git. Your `netGoat` project will be maintained using Git. Understanding the benefits of and using an RCS is something you should walk away from a computing major with.

note: Installing `git` Most Linux distributions have `git` available as a package. All you need to do is get `git` .
Debian (Ubuntu) Example: `# apt-get install git`

Prerequisites

- Department UNIX Accounts
- Team Name

Project Workspace

1. Creating & Sharing Project Workspace

   Perform these steps in one team member's home directory. Only files for this project should be placed in the project workspace.

   1. Create a collaboration directory for collaborative projects: `$ mkdir -p ~/collab/git`
   2. Change directory permissions so that all users have execute access to `~/collab` and `~/collab/git` (see `chmod(1)`)
   3. Change directory into the created directory (`~/collab/git`)
   4. Initialize the master `netGoat` project repository: `$ git init --bare repoName.git`
      You may choose an appropriately named repository name, but it shall end in `.git` (Git naming convention).
   5. Change directory into the created directory (`~/collab/git/repoName.git`)
   6. Share the project directory: (Copy, Paste, and Edit this command)

      ```
      $ setfacl -R -m
      d:u:instUser:rwx,d:u:user1:rwx,d:u:user2:rwx,d:g::x,d:o::x,u:instUser:rwx,u:user1:rwx,u:user2:rwx,g::x,o::x
      .
      ```

Replace *user1* with one team member's username, and replace *user2* with the other team member's username. Replace *instUser* with your instructor's username.
Note: Your instructor will have access to this repository as well; this is how your code will be submitted for `netGoat`, and how you can discuss coding questions with your instructor.

7. Verify project directory setup: `$ getfacl .`

   Output should look similar to:

```
# file: .
# owner: username
# group: mids
user::rwx
user:user1:rwx
user:user2:rwx
user:instUser:rwx
group::--x
mask::rwx
other::--x
default:user::rwx
default:user:user1:rwx
default:user:user2:rwx
default:user:instUser:rwx
default:group::--x
default:mask::rwx
default:other::--x
```

> **note: Deleting a FACL** If you want to delete a *File Access Control List* (FACL) from a file of directory run the following command:
> `$ setfacl -b -d fileSystemEntity`
> (see `setfacl(1)`)

If your output does not look similar to the above, delete the directory (`~/collab/git/repoName.git`) and start the process again.

Your project is now under revision control, and can be cloned and committed to by those who have access to the repository directory.

Cloning Repository

Each team member, including the team member who created the bare repositroy, and your instructor will clone the master repository. Each clone of the repository is a separate working copy of the repository. The whole purpose of the revision control system is to manage the changes between the various cloned repsoitories.

1. Configure Git:
   Note: Each team member should configure Git.
   1. Specify your name for Git: `$ git config --global user.name "Your Name"`
   2. Specify your email for Git: `$ git config --global user.email m20####@usna.edu`
2. Clone Repository:
   Note: Each team member should clone the initial bare Git repository. The team member who created the repository above should clone a copy of the repository elsewhere under their home directory.

   > **note: Many Clones** You can follow these same steps to have multiple clones of a repository. For example: there is one clone of the course repository on the department server, one on the course director's Mac workstation at home, and one on a USB drive.

   1. Create/navigate to the directory that you want the repository directory to be copied to.
      Note: Cloning a Git repository will cause a repository directory to be created in the current working director.
   2. Clone the remote Git repository:
      - Team member that created bare repository:
        Note: You will work from this cloned copy of the origin/master repository, you will not (and shall not) work from the repository in `~/collab/git/repoName.git`.
        1. Enter command: `$ git clone ~/collab/git/repoName.git`
      - Team member that did not create bare repository:
        1. Enter command: `$ git clone ssh://yourUsername@mope.academy.usna.edu/home/mids/createdRepoUsername/collab/git/repoName.git`
3. Verify repository cloning and remote target:
   1. Enter command: `$ git remote -v`
      Output should look similar to:

```
origin  .../home/mids/createdRepoUsername/collab/git/repoName.git (fetch)
origin  .../home/mids/createdRepoUsername/collab/git/repoName.git (push)
```

Git Commonly Used Commands

The following briefly describes the commonly used commands for Git. Further details can be found under the `git(1)` man page, the Git help system (`git help [subCommand]`), individual Git subcommand man page (e.g. `$ man git-subCommand`), or the Git online documentation.

- Adding to Repository:

Git's `add` subcommand is used to add files into the repository, the `add` sub command recursively searches directories adding all files and subdirectories. Git is generally friendly, in that if you forget a required argument Git will give you hints.

`git add <.>`
`git add <file1> ... [fileN]`
Note: The `add` command is relative to your current working directory not the root of the repository.
With Git you need to `add`, or stage, files to be included in a `commit`.

- Committing to Repository:

  You use the `commit` subcommand to have your changes committed to the local repository. You should get in the practice of making useful commit messages.
  `git commit -m "Added X. Updated Y."`

- Local Repository Status:

  You use the `status` subcommand to list the changes staged for commit, and list the unstaged changes of the local repository.
  `git status`

- Push to Remote Repository:

  You use the `push` subcommand to push updates from the local repository to the remote repository.
  `git push --all` **Note:** The first `push`/`pull` you do to/from a bare repository needs to include the `--all` option.
  `git push`

- Pull from a Remote Repository:

  You use the `pull` subcommand to pull updates from the remote repository to your local repository.
  `git pull --all` **Note:** The first `push`/`pull` you do to/from a bare repository needs to include the `--all` option.
  `git pull`

- See also: `git mv`, `git rm`

Required Completion Date: 1600 01 Mar 2018

[–] Milestone: 1 – Usage, Options, Help, & Error Reporting

In this project you will need to make use of functions, and make use of limited global variables. At future points within the project there will be times when you will send a signal to your program that is running and have it perform a task that your program will setup as the program starts, but you will not be able to provide any arguments to that task. If a milestone specification says to create a function, then you need to create and call a function. In general the use of functions in source code supports the Rule of Modularity, the Concept of Reusability, and the Concept of Abstraction.

Prerequisites

- Milestone 0 - Project Setup
- Basic C Programming
- Systems Programming Fundamentals: Systems Programming Concepts, Lab - Echo the Cat

Milestone Settings

As you add capabilities to `netGoat` the milestone settings may change, but the requirements of each milestone will not change. Your final `netGoat` will be tested on the latest settings.

- Minimum Number of Command Line Arguments: *None*
- `gcc` Required Compilation Flags: `-Wall`
- `gcc` Recommended Compilation Flags: *No Additional*

Usage

Your program will ultimately take command line arguments that alter the manner in which the program operates.

- If the minimum number of command line arguments is not entered, output a simple usage message.
- The code that outputs the usage message shall be in a stand alone function.
- You may use Descriptor I/O or Stream I/O to output the usage statement.
- The simple usage message shall include a statement about the help option.
- Exit the program without error after the help message is displayed.

Options

Your program will ultimately have numerous options and capabilities whose details are beyond the scope of a simple usage statement. Some options will take arguments, some options will not take arguments. Future milestones will specify further options that shall be implemented, you are encouraged to add additional options as you deem operationally relevant.

- Use `getopt` to handle the detection and identification of command line options.
- You may change option characters as the project goes on.

Help Option

The help option will display more detailed information about the options and capabilities of your program.

- Decide on a help option character.
- The code that outputs the detailed help message shall be in a stand alone function.
- You may use Descriptor I/O or Stream I/O to output the help message.
- For each option provide a write up that describes what the option does.
- Exit the program without error after the help message is displayed.

Error Reporting

In larger projects the initial extra effort put into making functions is easily justified by the gains to readability, reusability, and abstraction. Consitently formatted output, especially errors, is a component in designing programs that adhere to the Rule of Composition.

- Design an error reporting format that will be used throughout your code to report errors (program exits) and warnings (program continues).
- Design function(s) that implements your error reporting format.
- Your errors and warnings shall be written to Standard Error (`STDERR_FILENO` , *`stderr`*).
- Review `/usr/include/asm-generic/errno-base.h` and `/usr/include/asm-generic/errno.h` for standard system errors.
- Use your error reporting function(s) to report errors and warnings throughout `netGoat` .

Recommended Completion Date: 10 Mar 2018

[–] Milestone: 2 – Stray `cat`

Ultimately your `netGoat` program is going to be sending bytes between processes that are running on separate hosts. We don't have enough background to start the network communications, but that does not mean that there are not other parts of the problem that we can begin to solve.

Prerequisites

- Milestone 1 - Usage, Options, Help, & Error Reporting
- Systems Programming Fundamentals: Descriptor I/O, Lab - myCat

Milestone Settings

As you add capabilities to `netGoat` the milestone settings may change, but the requirements of each milestone will not change. Assumptions will carry across milestones; later milestones may negate earlier assumptions. Your final `netGoat` will be tested on the latest assumptions and settings.

- Minimum Number of Command Line Arguments: *None*
- `gcc` Required Compilation Flags: `-Wall`
- `gcc` Recommended Compilation Flags: *No Additional*
- Assumptions:
  - There will be at most one read from filename command line option.

Basic `cat` Functionality

Just like the `nc` utility, your `netGoat` will default to reading bytes from standard input and sending them to a process on a remote system; for now just output the bytes read from standard input to standard output. As you gain capabilities through class activities you will redirect the output of bytes away from standard output to other descriptors.

- After all command line arguments are handled appropriately, read in data from standard input until the End-of-File and output the data to standard output. Use Descriptor I/O to access data from standard input and output to standard output.

> **note: EOF** On UNIX systems the sequence `Ctrl-d` is used to signify the end of input, also known as *End-of-File* (`EOF` with Stream I/O).

- (Read from Filename) Add a new single character option that will take a single filename as a mandatory argument. Use Descriptor I/O to access the contents of the filename specified and output the contents of the file to standard output.
- Your solution shall handle arbitrary sized files.
- Upon reaching the End-of-File exit normally.
- If you are unable to access the data from either standard input or files specified as command line arguments, then output a descriptive error message, and exit the program returning an error.

Recommended Completion Date: 21 Mar 2018

[–] Milestone: 3 – Now You See Me, Now You Don't

Cyber Operations are much like yin-yang – defenders are driven by attacker actions, and attackers are driven by defender actions. Successful cyber operators are both capable defenders and attackers. Your `netGoat` will be capable of various actions, actions that can be used for defense (training) or offense (when directed). Ethical use of the code that you write is part of the moral character that makes you a Gray Hat, "With great power comes great responsibility" (*Voltaire*).

Similar to the Hiding from `ls` lab your `netGoat` will have the capability of hiding from `ls`; i.e. when activated your `netGoat` will not reside in the file system on disk while in execution. But unlike Hiding from `ls` your `netGoat` process may end from any number of states within your code and yet still need to be written back to disk before exiting, this will require some extra material not covered specifically through class discussions – Exit Handlers.

Prerequisites

- Milestone 2 - Stray `cat`
- File Systems: OS Internals, On Disk, Lab - Hiding from `ls`
- Processes: Process Life Cycle

Milestone Settings

As you add capabilities to `netGoat` the milestone settings may change, but the requirements of each milestone will not change. Assumptions will carry across milestones; later milestones may negate earlier assumptions. Your final `netGoat` will be tested on the latest assumptions and settings.

- Minimum Number of Command Line Arguments: *None*
- `gcc` Required Compilation Flags: `-Wall`
- `gcc` Recommended Compilation Flags: *No Additional*
- Assumptions:
  - Your `netGoat` will be started from a non-NFS portion of the file system on disk.
  - Your `netGoat` will not be terminated via a signal.
  - There will only be one reference on disk to your `netGoat`

Basic Hiding

While in *stealth* mode `netGoat` should not be visible to `ls`; i.e. the `netGoat` file should not exist in the file system on disk. **Note:** Recall to successfully test this option the `netGoat` executable needs to be in a non-NFS portion of the file system.

- Add a new single character option that does not take any arguments to signify that `netGoat` should enter stealth mode.

Basic Offline Persistence

Your `lsHiding` program hid a file from `ls` but wrote the file back to `/tmp` so that the file could be accessed after your program completed. One of the many desirable properties of malware is persistence. You will implement the most basic form of persistence combined with basic stealth; there are more advanced stealth techniques that are beyond the scope of SY204.

- If stealth mode is enabled, then before exiting write the `netGoat` executable back to the file system (current working directory) on disk using the name of the command. **Note:** See Exit Handlers for implementation ideas.
- If you are unable to setup basic offline persistence, then output an error and do not enter stealth mode, but continue program execution.

Recommended Completion Date: 31 Mar 2018

[–] Milestone: 4 – Standard Payloads

This milestone marks the first time you will choose between various options to implement as a part of your `netGoat`, this is where your `netGoat`s start becoming unique in their capabilities.

Prerequisites

- Milestone 3 - Now You See Me, Now You Don't
- File Systems: File Systems - On Disk
- Processes: Signals, Signal Handling, Lab - Baseball, Creating Processes

Milestone Settings

As you add capabilities to `netGoat` the milestone settings may change, but the requirements of each milestone will not change. Assumptions will carry across milestones; later milestones may negate earlier assumptions. Your final `netGoat` will be tested on the latest assumptions and settings.

- Minimum Number of Command Line Arguments: *None*
- `gcc` Required Compilation Flags: `-Wall`
- `gcc` Recommended Compilation Flags: *No Additional*
- Assumptions:
  - Use of standard ugo permissions is sufficient.
  - You WILL NOT be authorized to test *F Bob-omb* or *Seat's Taken* on department lab systems.
  - Your code will be tested on a default configured Ubuntu 16.04 LTS system.
  - Your code will be tested on an EXT4 file system.

You have chosen ... wisely

Choose two of the options below; have one action triggered by `SIGUSR1`, have the other action triggered by `SIGUSR2`.

- Add a statement in the usage message for each of the signal triggered events.

Peek-A-Boo

Toggle stealth mode upon receipt of the specified signal. Toggling should work regardless of whether the stealth option was entered via the command line or not.

- If stealth mode is not activated, then:
  - Activate stealth mode (`netGoat` not visible by `ls`).
  - Meet Basic Offline Persistence requirements
- If stealth mode is activated, then:
  - Deactivate stealth mode (`netGoat` visible by `ls`).
  - Disregard Basic Offline Persistence requirements

Me Too

A concern with `/tmp` is that being temporary the contents in `/tmp` may be deleted, for example: typically during system start up, after a set amount of time, or if space is needed. `/var/tmp` is persistent across reboots, but is available later in the boot cycle than `/tmp`.
Note: You may use `tempnam(3)` or another temporary file library (`mkstemp(3)`)

- Create a copy of `netGoat` in `/var/tmp`.
- Regardless of the `umask`, the copy in `/var/tmp` shall be readable and executable from any user account on the system, the copy shall be writable by the owner only (see `chmod(2)`).
  Note: DO NOT alter the `umask`.
- Each receipt of the signal shall create a new copy (uniquely named) of `netGoat` in `/var/tmp`.

F Bob-omb

Fork bombs are a way to attack Availability of the targeted system. Recall, that like other resources the process table is a finite size. When the process table is full new processes cannot be created. Ultimately a fork bomb causes a system to hang, typically requiring the system to be restarted.

- When triggered, fill the process (zombie or otherwise) table with processes.
  Note: It is recommended that you save your work before testing this feature, regardless of the system that you test it on.
  Note: The lab systems are protected from this type of Availability attack; it is likely that your VM is not.

Seat's Taken

Just like the process table, the i-node table of a UNIX file system is a finite size. When the i-node table is full no new files can be created in that file system, at least on disk. Many processes will continue to operate, but new files (such as log files) will not be able to be created. Depending on how a system is configured servers may continue to run without the ability to log events, routine or otherwise (malicious).

in operations: Testing Malware Malicious software (malware) is best tested in an environment that can be easily reset or rolled back to a known good state, your Ubuntu VMs are a good place to develop and test your offensive operations for a number of reasons:

- You own them, by definition, you have the authority to test code on your VMs. Remember, you should never employ offensive cyber operations on any information system without the express written consent of the information system owner.
  Note: That being said, if a system is known to be infected DO NOT connect that system to a non-test environment. Specifically, DO NOT connect that system to a government or production network, such as the USNA Enterprise Network.
- VMs are easy to snapshot and revert to a known good configuration (know yourself).

That being said, some more advanced malware packages look for file system relics that indicate that the system is in fact a VM, and the malware alters its behavior. The idea being that the general worldwide user population does not run virtual machines, but security researches do run [many] virtual machines; therefore if the malware is on a virtual machine it is likely on a system that is being monitored. But then malware that doesn't attack or propagate across virtual machines makes virtual machines safer against the malware's designers from compromising the virtual machines.
Attackers assess and mitigate risk just as defenders assess and mitigate risk.

- When triggered, fill the i-node table with file system entities.
  Note: It is *highly* recommended that you verify that the `/tmp` directory on your test system does in fact clear the contents of `/tmp` on reboot. The default configuration for Ubuntu 16.04 LTS is to clear the contents of `/tmp` on reboot.
  Note: It is recommended that you create file system entities in `/tmp` to facilitate easy system recovery/restoration
  Note: Different file systems have different characteristics, the method you fill File System X may not be the same way you fill File System Y.

Recommended Completion Date: 07 Apr 2018

[–] Milestone: 5 – Nothing Can Stop Me Now or Variable Payloads

This milestone is a major option point, choose one of the two options to implement.

Prerequisites

- Milestone 4 - Standard Payloads
- Processes: Creating Processes, Program Execution

Milestone Settings

As you add capabilities to `netGoat` the milestone settings may change, but the requirements of each milestone will not change. Assumptions will carry across milestones; later milestones may negate earlier assumptions. Your final `netGoat` will be tested on the latest assumptions and settings.

- Minimum Number of Command Line Arguments: *None*
- `gcc` Required Compilation Flags: `-Wall`
- `gcc` Recommended Compilation Flags: *No Additional*
- Assumptions:
  - No Additional.

Persistence or Load Up

For this milestone you will either add more persistence to your `netGoat` through a heartbeat mechanism, or add the capability to set the actions for `SIGUSR1` and `SIGUSR2` via command line options. Implement either *Nothing Can Stop Me Now* or *Variable Payloads*.

Nothing Can Stop Me Now

Implement a simple heartbeat mechanism using `SIGUSR1` as the heartbeat signal between two processes: one heartbeat monitor process, one heartbeat sender process. This will require you to change the disposition of `SIGUSR1` to a routine other than one of the payloads from Milestone 4. The process who is monitoring heartbeats will only be concerned with monitoring heartbeats and creating a new process if heartbeats are not being received. The process being monitored will perform the primary functions of `netGoat`.

- After all options have been configured, have the initial process create a grandchild process.
  - The grandparent process will monitor heartbeat signals from a grandchild process.
  - A parent process will only serve the purpose of creating a grandchild process.
  - A grandchild process will perform core `netGoat` functionality (as specified from the original command line arguments) and send heartbeat signals to the grandparent process.
- A grandchild process shall send a heartbeat signal (`SIGUSR1`) to the grandparent process every 60 seconds (see `alarm(2)`).
- If the grandparent process does not receive a heartbeat signal from a grandchild process within 180 seconds (see `clock_gettime(2)`), then the grandparent shall create a new set of parent and grandchild processes for monitoring.
- Add statements in the usage message that explain how the heartbeat mechanism functions.
  Note: The heartbeat feature of your `netGoat` will always run, this feature will not be configured by a command line option/argument.

Variable Payloads

Add additional command line options to set the triggered actions for `SIGUSR1` and `SIGUSR2` from all four of the options from Milestone 4.

- Implement the two options from Milestone 4 - Standard Payloads that you did not initially implement to increase your payload options.
- Add a command line option to set the Standard Payload action that will be triggered when `SIGUSR1` is received.
- Add a command line option to set the Standard Payload action that will be triggered when `SIGUSR2` is received.
- Add statements in the usage message that: explain what the additional command line options are and how to specify a payload to associate with the signal, explain what the various payload options are, and explain what the default payloads are
  - You are free to chose any two options as default options
  - It is not an error if the same payload is set for `SIGUSR1` and `SIGUSR2`

Recommended Completion Date: 14 Apr 2018

[–] Milestone: 6 – Plumbing 101

You are nearing the core functionality of `nc(1)` (communicating between processes on different hosts), but we are not quite there yet. We still need to explore networked communications in C, but we there is other work we can do with the tools we have so far.

Due to security concerns most `nc(1)` implementations due not include the `-c` or `-e` options from v1.10 of netcat (see *Wikipedia - Netcat*). Your netGoat will have similar capabilities regarding netcat's execute (`-e`) option. Your `netGoat` will essentially have the capability to redirect data from one descriptor to the standard input of a another process that the initial `netGoat` created, and redirect the standard output from the other process to a descriptor. The process created will change it's process image to a new process image.

in operations: `/bin` **Backdoors** Operators (defenders and attackers) need to know their toolsets and how to use those toolsets. Capable cyber operators are comfortable with the command line and the set of tools that are commonly found on systems. Simple utilities can be chained together to solve complex problems. Take for example a stripped down version of `nc(1)` , one that does not support `-c` or `-e` ; we ask ourselves, "Can we get the functionality of `-e` another way?" The answer is *yes*, there are ample tools in `/bin` that allow a user to setup a backdoor on a system.
Netcat is in `/bin` , so we have a way to do basic communications across a network. Shells are in `/bin` , so we have a way to start a shell. The last piece we need is a way to connect the inputs and outputs of a netcat process and a shell process, we need a pipe in the file system. We can use `mkfifo(1)` in `/bin/usr` or `mknod(1)` in `/bin` to create a named pipe in the file system. So even with a stripped down version of netcat, a backdoor can be created on the system with a series of two shell commands, assuming you can get the two commands to execute in the first place.

note: You DO NOT have permission to circumvent the security posture of the department systems (lab machines or otherwise) for prolonged periods of time. You SHALL NOT leave a session (local login or remote login) on a department system unattended while you are testing the combination of Milestone 6 and Milestone 7.

Prerequisites

- Milestone 5 - Nothing Can Stop Me Now or Variable Payloads
- Inter-Process Communication: I/O Duplication, Pipes

Milestone Settings

As you add capabilities to `netGoat` the milestone settings may change, but the requirements of each milestone will not change. Assumptions will carry across milestones; later milestones may negate earlier assumptions. Your final `netGoat` will be tested on the latest assumptions and settings.

- Minimum Number of Command Line Arguments: *None*
- `gcc` Required Compilation Flags: `-Wall`
- `gcc` Recommended Compilation Flags: *No Additional*
- Assumptions:
  - No Additional.

Dry Fit

Try the following specifications on for size, once you are able to communicate across a network portions of the below specifications will change.

- (Executive Decision) Add an additional command line option that takes a mandatory argument, the mandatory argument is the command that should be executed.
  - The mandatory argument will *only* be the name of the command to be executed; i.e. the command to be execute will not include any command line arguments.
- Once other command line options are handled, but before you read or write data to or from a descriptor or file (see Milestone 2 - Basic `cat` Functionality), create a new process that will execute to become the command named from the command line option.
  - The command will be executed with no command line arguments.
  - The execute call should work if the command name does not include a `/` character.
- Create pipes as needed to have two way communication between the parent process (`netGoat`) and the child process (executed command).
  - (For Milestone 6) Data read from `netGoat` 's standard input shall be redirected to the standard input of the executed command.
  - (For Milestone 6) Data written to the executed command's standard output shall be redirected to the standard output of `netGoat` .
- When the child process terminates, `netGoat` shall gracefully exit.

Secure the Path

Attackers take defensive measures when operating and designing exploits, so should you.

- If the *PATH* environment variable is unset when your program starts, then set the *PATH* environment variable to only search standard binary locations in this order: `/bin` , `/usr/bin` , `/usr/local/bin` .

Recommended Completion Date: 21 Apr 2018

[–] Milestone: 7 – These are the Packets You're Looking For

In this milestone you add the core capability to `netGoat` , communications between two different hosts across a network. Additionally, you get to decide on the last customization option to your `netGoat` . After completing Milestone 7 you will have a capable tool that you can use ethically.

Prerequisites

- Milestone 6 - Plumbing 101
- Inter-Process Communication: Sockets – Introduction, Sockets – TCP/IP Fundamentals, Sockets – Internet Domain (Part 1), Sockets – Internet Domain (Part 2).

Milestone Settings

As you add capabilities to `netGoat` the milestone settings may change, but the requirements of each milestone will not change. Assumptions will carry across milestones; later milestones may negate earlier assumptions. Your final `netGoat` will be tested on the latest assumptions and settings.

- Minimum Number of Command Line Arguments: *Two* (client), *Two* (server [command line option with mandatory argument), *One* (Phone Home [command line option with no argument])
- `gcc` Required Compilation Flags: `-Wall`
- `gcc` Recommended Compilation Flags: *No Additional*
- Assumptions:
  - No Additional.

Socket To Me (Soldering Required)

Netcat is considered by many to be a swiss army knife, its blade is the ability to send raw data via TCP or UDP. Your `netGoat` will have the same basic blade as netcat. You will add functionality for your `netGoat` to act as either a client or server, your `netGoat` only needs to communicate using TCP/IPv4.

The client mode of operation will be the default mode `netGoat`, no option required. In client mode your `netGoat` will initiate a connection to a remote host. Your `netGoat` will read data from standard input and send that data to the remote host.

- The first non-option command line argument, shall be used as the fully qualified domain name of the remote host to connect to.
- The second non-option command line argument, shall be used as the port number of the remote host to connect to.
- Cycle between: sending data read from standard input across the socket, and writing data read from the socket to standard output.
- Once connected, if the Read from Filename option (Milestone 2) was specified, send the data from the file before sending data read from standard input.
- If the connection to the remote host fails, then output an error message and exit with an error.

In server mode, your `netGoat` will listen for an incoming connection request from a remote host. Your `netGoat` will read data from standard input and send that data to the remote host.

- Add an additional command line option that takes a mandatory argument, the mandatory argument is the port that `netGoat` shall listen for an incoming connection on.
  - `netGoat` can either be in client mode or server mode. If there are non-option command line arguments and server mode is specified, then output an error message and exit with an error.

For both client and server mode of operation:

- Once connected, if the Executive Decision option (Milestone 6) was specified:
  1. Send the data from the socket to the executed command's standard input.
  2. Send the data from the executed command's standard output to the socket.

Your Call Cannot be Completed as Dialled

Many utilities require a minimum number of arguments, and report usage messages when called incorrectly.

- If the incorrect number of arguments to enter client mode or server mode are not entered, then output a usage message, and exit with an error.

Call Back or I Love my Tacks

For this milestone you will either add reverse call capability or the expand the execute command capability to include command line arguments. Implement either *Phone Home* or *Commands with Controls*.

Phone Home (Best Done with a Speak & Spell)

In phone home mode `netGoat` will act like client mode, except that the remote host and remote port to connect to will be hard coded in the `netGoat` instead of being entered via command line arguments.

Commands with Controls (Path to Pivots)

Executing a command without arguments has its limitations. Alter Milestone 6 – Dry Fit to support executing a command with command line arguments.

Required Completion Date: 25 Apr 2018

[+] Milestone: 8 – Witness the Power of this Fully Armed and Operational `netGoat`

[–] Milestone: Oral Presentation – This is Our Jeep

Oral Presentation

As Officers and professionals in the cyber domain you need to be able to convey ideas orally. In the oral presentation each group member will have speaking parts. The oral presentation should be targeted for 7 minutes, not including questions by other students or the instructor. You will be given 10 minutes total: setup, presentation, questions.

Outline

Oral Presentation Outline:

> **note: The Mother of All Demos** When you fully grok a subject, you are able to create extraordinary demonstrations. At a minimum review the highlights from Doug Engelbart's *The Mother of All Demos* from 1968; you are encouraged to view the full demonstration at least once during your time at the Naval Academy.
>
> - Engelbart, Doug. "The Mother of All Demos (Highlights Playlist)." 1968. *YouTube*.
> - Engelbart, Doug. "The Mother of All Demos (Full)." 1968. *YouTube*.

0. Setup (not counted in 7 minute target time)
1. Team Introduction

   Introduce the team members, and group name.

2. Overview of `netGoat` Options Chosen (Milestone 4, Milestone 5, Milestone 7)

   A big picture view of your `netGoat`, specifically mention the options your group decided to implement.

3. Milestone 3
4. Milestone 4
5. Milestone 5
6. Milestone 6
7. Milestone 7
8. ~~Milestone 8~~
9. Lessons Learned

   Discuss at least two lessons learned, things you would have done differently given 20-20 hind sight. The lessons learned can be either technical related (use system call X), group interaction related, or general project management related.

10. Questions

Format

Adhere to the following format specifications:

- Submit Method:
  - (Electronic/Soft) Committed and pushed to team's master repository
  - (Print/Hard) Turned into Instructor at time of presentation
- Submit Formats: PDF (display); PowerPoint/LaTeX (source) or Skit Script (source)

Milestone Discussion Points

For each of the milestones discuss the following:

- General What/Why:

  Discuss what decision points you made and why you came to that decision. This applies to big decisions like Milestone 4/Milestone 5.

- Implementation:

  Briefly walk-through your code for the milestone. Discuss your approach to solving the problem. Discuss any challenges you faced, and how you overcame those challenges.

- Technique Demonstration

  Demonstrate basic usage of your `netGoat`. Discuss how you envision your `netGoat` being used either in support of an offensive or defensive action; i.e. create and discuss a vignette that includes the use of your `netGoat`, a vignette that ties the various miletstones together.

Focus Areas

It is expected that you will spend more time discussing the later milestones than the earlier milestones (recall the 7 minute target). The core features of `netGoat` come as you start combining the different features together, specifically communicating across a network.

Speaking Roles

Each team member shall have a speaking role, specifically beyond the Team Introduction and Overview the speaking roles shall cycle among group members for each of the oral presentation outline sections. Each group member shall be able to speak to all of the milestones, aspects of `netGoat`.

Rubric

- Oral Presentation Rubric

Required Completion Date: In Class 02 May 2018

[–] Milestone: Written Report – We Pimped Our Jeep

Written Report

As Officers and professionals in the cyber domain you need to be able to convey ideas via written forms of communication. Your written report will expand upon topics from and cover topics not in the oral presentation.

Outline

Written Report Outline:

Your written report will contain the following sections:

1. Title Page
2. Table of Contents
3. Introduction
4. Design Decisions
5. Procedures
6. Malware Characteristics
7. Techniques
8. Lessons Learned
9. Conclusion
10. Appendices:
    A. References
    B. Source Code
    C. Presentation Slides

Format

Adhere to the following format specifications:

- Submit Method:
  - (Electronic/Soft) Committed and pushed to team's master repository
  - (Print/Hard) Turned into Instructor by due date
- Submit Formats: PDF (display), Word/LaTeX (source)
- Length:
  Note: Not including: Title Page, Table of Contents, Appendices
  - Minimum: 5 pages
  - Expected: 5+ pages
  - Maximum: 12 pages
- Font:
  - Title Style: sans serif font
  - Title Size: variable, no larger than 20 pt
  - Body Style: serif font, Times New Roman preferred
  - Body Size: 12 pt
  - Justified: left
  - Kerning: yes
  - Code Style: monospace, Lucida Console preferred
  - Code Size: 10 pt
  - Code Misc: indent/space code for readability, spacing in report does not need to match spacing in source code

- Line Spacing:
  - Minimum: 12 pt
  - Maximum: 1.5 lines
- Page Numbering:
  - Location: footer, centered (all page numbers)
  - Title Page: unnumbered
  - Table of Contents Style: lower case Roman Numberals
  - Table of Contents Start: i
  - Main Body: Arabic Numerals
  - Main Body Start: 1
  - References: *Appendix*-Arabic Numerals; e.g. A-1
  - Source Code: Arabic Numerals
  - Presentation: per slide number, one slide per page

Introduction

Introduce the reader to what your `netGoat` does. Discuss the general purpose of `netGoat` as well as the features that you implemented beyond standard `nc`.

Design Decisions

For each milestone, beginning with Milestone 1, discuss what features you implemented, how you implemented the features, and why you chose to implement the feature in that way; discuss the details. Details to include small decisions such as the characters you chose for the command line options. Go beyond "just because" or "it seemed like a good idea". No matter how banal the decision, there was a reason (likely valid) that led you to that decision; i.e. discuss your reasoning behind the decisions you made. Discuss the implementation challenges you encountered and how you overcame those challenges.

Procedures (`$ man netGoat`)

Discuss and present examples of how to use the features of your `netGoat`. Begin with the usage statement and help options, and go through all of the options that you implemented. Include command line examples, similar to examples in lab assignments. Discuss error conditions that your `netGoat` detects and how it reacts to those errors (recover, report and recover, report and exit). Include detailed error statements and any exit values returned.

Malware Characteristics (Snarling Beast Guy)

Discuss how your `netGoat` is similar to various types of malware (virus, trojan horse, etc.).

Techniques

Discuss how your `netGoat` could be used in operations. You should state that `netGoat` is installed on the system as `netGoat` or another named binary. If you do not explicitly state what `netGoat` is named as, the name will assumed to be `netGoat`.

Discuss at least one vignette that utilizes `netGoat` in a defensive way, from the viewpoint of the target. You may assume that the defender installed `netGoat` on the system as `netGoat` or another named binary.

Discuss at least one vignette that utilizes `netGoat` in an offensive way, from the viewpoint of the target. You may assume that the attacker exploits a vulnerability using methods/tools outside of `netGoat`.

Lessons Learned

Discuss at least five lessons learned (good, bad, other; at least one each topic area). For each lesson learned discuss: the issue/point, provide an example, recommendation(s) for future project. Discuss the lessons learned from your, the student's, viewpoint.

Technical related.

Project Management related.

Group Interaction related.

Conclusion (`exit(0);`)

Forth coming.

Rubric

- Written Report Rubric

Required Completion Date: 1200 04 May 2018

[–] Milestone: Peer Evaluation

Peer Review

You will rate each other group member using the Peer Evaluation Form.

Required Completion Date: 1200 04 May 2018

++

None.