UNIVERSITY OF WATERLOO
Cheriton School of Computer Science

CS 458/658 **Computer Security and Privacy** **Winter 2015**
Urs Hengartner

ASSIGNMENT 2

Assignment due date: **Monday, March 9th, 2015 3:00 pm**

Total marks: 66 (+ 4 possible bonus)
Written Response Questions TA: Hassan Khan
Programming Question TA: Cecylia Bocovich

Please use Piazza for all communication. Ask a private question if necessary. The TAs' office hours are also posted to Piazza.

Written Response Questions [26 marks]

Note: For written questions, please be sure to use complete, grammatically correct sentences. You will be marked on the presentation and clarity of your answers as well as the content.

1. [10 marks] **Bell-La Padula Confidentiality Model / Biba Integrity Model.**
   In the Bell-La Padula confidentiality model, consider a user Alice with clearance level (Secret, {CIA, FBI}), and five documents with the following classification levels:

   - D101 has level (Top Secret, {NSA, CIA, FBI}),
   - D102 has level (Secret, {FBI}),
   - D103 has level (Classified, {NSA, CIA, FBI}),
   - D104 has level (Secret, {CIA, FBI}), and
   - D105 has level (Unclassified, {CIA}).

   (a) [5 marks] Which of these documents can Alice read? Which can she write to?

   (b) [5 marks] Now assume a *dynamic* version of the Biba integrity model that incorporates a suitable version of the low watermark property, where the updates involve the *glb* (greatest lower bound) function. Now suppose Alice performs the following five actions (in the given order):

      i. Alice writes to D105,

ii. Alice writes to D104,

iii. Alice reads from D103,

iv. Alice writes to D102, and

v. Alice writes to D101.

After each of Alice's above actions, specify the integrity level of Alice *and* of the document she just accessed.

Note: You should use the following dominance hierarchy:

$$\textbf{Unclassified} \leq \textbf{Classified} \leq \textbf{Secret} \leq \textbf{Top Secret}.$$

2. [10 marks total] **Intrusion Detection Systems.**
Microsoft's Internet Information Server (IIS) has a filtering mechanism to prevent maliciously crafted URLs from breaking out of a directory. For example, a URL such as

"`http://foo.com/scripts/../../winnt/system32/cmd.exe?/c+dir`"

tries to break out of the scripts directory and to execute the `cmd` command located in `c:\winnt\system32`. However, an old version of IIS performed this filtering *before* unicode expansion, and was thus susceptible to a "unicode attack" wherein a URL such as

"`http://foo.com/scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir`"

would bypass the filtering mechanism, but ultimately get expanded to a URL that is identical to the malicious URL above.

To detect this kind of attack, Snort, a popular open-source intrusion detection system included a signature that looked for the pattern "`/..%c0%af../`" in HTTP packets.

(a) [3 marks] In normal usage, why should this signature have low false positive rate?

(b) [3 marks] Suppose an attacker discovers that a particular website is running the old version of IIS, which is vulnerable to the unicode attack, but that the website is protected by Snort (with the signature given above enabled). What might a clever attacker to do make a naive system administrator disable the Snort signature, thus making the vulnerability accessible to the attacker (1 mark)? How should the attacker avoid having his own IP address noticed while he carries out this attack?

(c) [2 marks] In light of your proposed attack from part (b), what can you conclude about the false positive rates advertised for security software like antivirus and IDS?

(d) [2 marks] How might the attacker undetectably exploit the IIS bug even if the Snort rule remains in place? .

3. **[8 marks] Firewall**

You are setting up an ecommerce website for a local entrepreneur. The web site is hosted on its own server, and will interact heavily with a database that is hosted on a second server. The database server also serves a number of internal users, as well as applications on the desktop computers of company employees.

(a) [3 marks] If there is only room in the budget for a single firewall, where should you place it (1 mark), and why (2 marks)?

(b) [3 marks] If more room is found in the budget to purchase a second firewall, where should you place it (1 mark), and why (2 marks)?

Programming Question [40 marks]

Background

Your corporation has recently been the victim of several attacks that have evaded your firewalls, and the administration would like to be alerted immediately when new attacks are launched. You have been tasked with the development of a custom intrusion detection system (IDS). This IDS will be network-based, and will monitor traffic for known attacks based on provided signatures. Your application will receive input from a network monitoring program and output any detected attacks, as well as some details about them. The output from your program will be used by other scripts to send alerts to the network administrators, or to dynamically add rules to the firewall, as deemed appropriate.

Please read the TCP/IP primer at the end of this handout if you are not familiar with TCP/IP.

Application Description

The host machine for the IDS will monitor network traffic using the `tcpdump` utility for Unix-like operating systems. `tcpdump` parses data packets and outputs human-readable representations. The exact command used to generate the packet information is:

```
tcpdump -vvv -XX -n -l
```

This command prints verbose information about packets, along with their contents, and does not convert IP addresses and ports into names. Output from this command will be piped into the standard input of your program. You will need to examine the data to determine if any attacks have occurred. Signatures for the attacks that should be detected are provided in the following sections. When an attack is detected, you will need to print an alert to the standard output stream. Alert messages should conform exactly to the following format:

```
[attack]: details
```

In your output, both *attack* and *details* should be replaced with the information specified in the relevant section of the assignment. The output from your program will be piped into a series of scripts written by another programmer. These scripts will react to your alerts in a manner determined by the network administrators. For this reason, it is important that you output alerts immediately when attacks occur; to do this, you must flush your standard output buffer immediately after writing a line. In summary, your IDS program is intended to be invoked in the following manner:

```
tcpdump -vvv -XX -n -l | ids | alert-script
```

Note that the `alert-script` file would be present in the hypothetical scenario described in the background for this question, but this script is not actually present in the testing environment.

Testing Environment

For each attack, a file containing an example of the attack has been provided. To test your IDS, you can write the contents of the sample file to the standard input of your program using the following command (be sure to replace *sample-file* with the actual name of the sample file):

```
ids < sample-file
```

Each of the sample files comes along with the expected output for your script. To ensure that your output is the expected one for the problem, you can compare the attacks that your program detected against the expected attacks using `diff`:

```
ids < sample-file | diff - expected-output-file
```

If your IDS is working properly, the output from this command should be empty. However, you should ensure that you are identifying attacks using the requested methodology (e.g., hard-coding output for the sample files is not a valid solution). Upon submission, your program will be tested using additional sample files.

[6 marks] Spoofed packets

Packets with spoofed sources or destinations are usually part of an attack, such as a distributed denial of service attack. In fact, this problem was common enough to prompt a best current practice entry (BCP 38). Network Ingress Filtering restricts outgoing network traffic from invalid source IP addresses. Your IDS can monitor all of the network traffic on your corporate LAN, where local computers are within the `10.97.0.0/16` IP range (i.e., every corporate computer has an IP address of `10.97.*.*` where each `*` is a value from 0 to 255). Every packet visible to your IDS is expected to be coming from or traveling to one of these local machines. Write a rule for your IDS that detects packets that have a clearly forged source **or destination** IP address.

- Value for *attack* in your output: `Spoofed IP address`
- Value for *details* in your output: `src:`*source*`, dst:`*destination* where *source* and *destination* are the IP addresses and ports from `tcpdump`

- Sample input for testing: `atk1-spoofed.log`
- Expected output for testing: `atk1-spoofed-output.log`

[6 marks] Unauthorized access

GeoIP is an effort to map IP addresses to geographic locations. Your corporation allows employees to work from home and ssh into their workstations remotely. However, your company policy does not allow any remote access from IP addresses outside of the Kitchener-Waterloo area. Connections coming from Kitchener or Waterloo and connections between two computers on the LAN are acceptable and should be ignored by the IDS.

A copy of a simplified version of MaxMind's GeoLite City database for Ontario has been provided with the assignment. Your IDS should read the `GeoLiteCity.csv` file at startup and raise an alert for incoming server connections from non-local IP addresses. Refer to the following page for information on GeoIP: http://dev.maxmind.com/ Note that we have modified the structure of the downloadable databases for the purpose of the assignment.

Write two rules for your IDS:

1. Detect when a remote computer (i.e., one outside of Kitchener-Waterloo or outside of the `10.97.0.0/16` IP range) attempts to connect to a server running within the LAN
   - Value for *attack* in your output: `Attempted server connection`
   - Value for *details* in your output: `rem:`*`remote`*`, srv:`*`server`* where *remote* and *server* are the IP addresses and ports of the external computer and the LAN-based server, respectively
2. Detect when a server running within the LAN accepts a connection from a non-local computer
   - Value for *attack* in your output: `Accepted server connection`
   - Value for *details* in your output: `rem:`*`remote`*`, srv:`*`server`* where *remote* and *server* are the IP addresses and ports of the external computer and the LAN-based server, respectively

For the purposes of this question, you should focus on TCP/IP connections. You may find the following page, which describes the process for TCP connection establishment, useful when designing your rules:
http://www.inetdaemon.com/tutorials/internet/tcp/3-way_handshake.shtml

You may also find this information on IPv4 address subnetting useful:
http://en.wikipedia.org/wiki/IP_address#Classless_subnetting

- Sample input for testing: `atk2-servers.log`
- Expected output for testing: `atk2-servers-output.log`

[6 marks] Known malicious hosts

Since connections to servers running within the LAN are easily detectable, many viruses receive commands by making outbound connections to the Internet instead. For example, a virus might connect to a website operated by the virus author in order to download new commands. Your IDS should detect any DNS queries for known malicious domains. Your IDS should read the `domains.txt` file provided with the assignment when starting up. This file contains 100 domains names, one per line, that are known to host malicious content. Your IDS should raise an alert if a DNS query for any of these domains is issued.

- Value for *attack* in your output: `Malicious host lookup`
- Value for *details* in your output: `src:`*`source`*`, host:`*`host`* where *source* is the source IP address and port of the machine performing the DNS query, and *host* is the malicious host whose IP address is being requested
- Sample input for testing: `atk3-hosts.log`
- Expected output for testing: `atk3-hosts-output.log`

[6 marks] Random Scanning

The Slammer worm infected over 75,000 hosts, and doubled in size every 8.5 seconds - making it the fastest spreading worm in history. Random scanning worms propogate quickly by launching attacks from infected machines on randomly selected IP addresses. To detect these scans, you will need to store some information about the activities of computers over a short period of time. An alert should be raised if you observe a single IP address sending a single UDP or TCP-SYN packet to 10 different IP addresses within a 2 second period. For simplicity, issuing multiple alerts for an ongoing scan is acceptable. Alerts should be raised **as soon as any of the following attacks occur** within a 2 second period:

1. **UDP attack**: The attacker sends a single UDP packet to at least 10 different IP addresses

2. **TCP attack**: The attacker sends a TCP-SYN packet to at least 10 different IP addresses

Note that, when testing your IDS with the sample file, all of the lines will appear to arrive at the same time since reading the file is nearly instantaneous. You should base your timing on the timestamps received from `tcpdump` rather than the system clock.

- Value for *attack* in your output: `Potential random scan`
- Value for *details* in your output: `att:`*`attacker`* where *attacker* is the IP address of the machine issuing the scans
- Sample inputs for testing: `atk4-scan-udp.log`, `atk4-scan-tcp.log`
- Expected output for testing (in all cases): `atk4-scan-output.log`

[6 marks] Code Red

One of the most iconic computer worms in history was the Code Red worm, released in July 2001. This worm scanned the Internet for vulnerable web servers running the Microsoft IIS web server. It exploited these servers using a buffer overflow exploit similar to the one you wrote in assignment 1. Upon successful exploitation, the worm would deface the website to display a message from the worm authors. Your IDS should detect attempts by the Code Red worm or variants (such as its successor, Code Red II) to exploit this IIS vulnerability.

More information about the worm and its attack is available on Wikipedia:

https://en.wikipedia.org/wiki/Code_Red_(computer_worm)

Write a rule for your IDS that detects TCP packets to port 80 (used for web traffic) containing a request that exploits the IIS vulnerability. You will need to read the contents of the packet from `tcpdump` to search for the attack. Ensure that your rule does not result in false positives for ordinary web traffic.

- Value for *attack* in your output: `Code Red exploit`
- Value for *details* in your output: `src:`*`source`*`, dst:`*`destination`* where *source* and *destination* are the source IP address and port information from `tcpdump`
- Sample input for testing: `atk5-codered.log`
- Expected output for testing: `atk5-codered-output.log`

[6 marks] Conficker

Many forms of malware make use of domain name generation to receive instructions without detection from traditional blacklisting techniques. The Conficker worm was one of the earliest forms of malware to use this technique. The following pages contain useful information about Conficker's behaviour:

- http://mtc.sri.com/Conficker/
- https://www.honeynet.org/files/KYE-Conficker.pdf

Write a rule for your IDS that detects the initial attempt of a Conficker worm (variant A) to communicate. If this connection to the worm authors can be identified, the scripts reading the output of your IDS could add a firewall rule to isolate the infected computer from the Internet. This will prevent the Conficker worm from spreading to other machines.

Hint: Use the information in the articles to expand your solution to the "known malicious hosts" attack. It is possible to accurately detect Conficker's communications without false positives.

Due to the nature of this question, sample input files for Conficker are only valid for one day. Thus, no sample input has been provided with the assignment. Instead, a page with additional information about Conficker and an option to generate a sample input file for a given day has been provided. You are urged to use the following site when answering this question:
https://olten.uwaterloo.ca/cs458/conficker

Note that the scripts on the aforementioned site will be unavailable during marking time (i.e., do not query the site as part of your solution). Additionally, the system time will be set to a random date when your IDS is tested.

- Value for *attack* in your output: `Conficker worm`
- Value for *details* in your output: `src:`*`source`* where *source* is the IP address and port of the machine infected with the Conficker worm
- Sample input for testing: See above
- Expected output for testing: `atk6-conficker-output.log`

[4 marks] Output requirements

- (2 marks) Ensure that your output conforms to the given format. In particular, ensure that there are no differences between your output and the expected output for the sample files.
- (1 mark) Ensure that you **flush the standard output buffer** after printing an alert so that it can be processed immediately. For example, in C you would write `fflush(stdout)`. This allows you to print alerts without waiting for another packet to arrive.
- (1 mark) Your program should exit gracefully when EOF is encountered in standard input.

[4 bonus marks] Bypassing the IDS

Many network-based intrusion detection systems can be avoided by employing some simple tricks. It is difficult for an IDS to robustly defeat these techniques due to fundamental limitations in the knowledge available to the system. The following paper describes several ways to carry out attacks while avoiding detection by an IDS:

http://insecure.org/stf/secnet_ids/secnet_ids.pdf

Complete the following tasks in your written submission:

1. Explain how the IDS rules for detecting network scans could be abused to frame an innocent system on the LAN.
2. Explain one way that the Code Red worm could avoid detection by your IDS by performing an evasion attack. Be specific when describing the attack.

Testing on the Ugster Machines

Once you are confident that your IDS is working correctly for the sample files, you can optionally use the *ugster* machines to perform several real-world tests. Using the same account credentials that you used for assignment 1, log into your designated ugster machine. Once you have logged into your ugster account, run `uml` to launch the virtual Linux environment for assignment 2. Recall that you should log in as `user` to interact with the machine, and `halt` when you want to halt the virtual environment. The virtual machines contain `domains.txt` in the user's home directory.

Once you have started your virtual machine, you will see several lines similar to these:

```
Virtual console # assigned device '/dev/pts/#'
```

Make note of the numbers at the end of these lines. You can use this information to open a second window into the uml. To do so, make a second SSH connection to the ugster while still keeping your original window open. You can now connect this second window to the uml by executing `screen /dev/pts/#`, where # is replaced with the value displayed earlier. After executing this command, the window will initially be blank. Press enter to gain access to a login prompt. To shut down this second screen, press `ctrl+a` followed by `k`.

For testing, you can launch your program using `tcpdump -vvv -XX -n -l |` *`ids`* in one window, while performing internet operations that you expect to trigger your IDS in the other. There are several alerts that you should be able to trigger with benign internet traffic. You can perform real-world tests for the following alerts:

Known malicious hosts: Execute `/usr/local/share/tests/test hosts`

Potential random scans: Execute `/usr/local/share/tests/test` *`METHOD`* where *METHOD* is one of `udp` or `syn`.

Conficker: No script is provided for testing this alert, but it is possible to test.
Hint: You may find the `nslookup` command helpful.

No scripts are provided for testing spoofed packets, unauthorized access, or the Code Red worm due to logistical issues. You should rely on the sample logs provided with the assignment for testing these alerts. **Do not attempt to test your IDS by running actual attacks against other systems.**

Evaluation

- For marking, we will compile and execute your IDS in the `/share` directory in a virtual machine.
- Your IDS will be executed inside a virtual machine with standard input provided by `tcpdump`. You can assume that your program will be executed from the current working directory, and that this directory will also contain `domains.txt` as well as `GeoLiteCity.csv`.
- Your IDS rules should not produce false positives. For example, raising an alert that Conficker has been detected whenever you receive any packet, irrespective of the contents of the packet, is not a valid solution.
- All relevant packets will follow the formats observed in the sample input files. If your program encounters a packet with an unknown format, it should be gracefully ignored.

Programming Languages

We have installed several of the most popular programming languages within the virtual machine. You may choose any language supported by the virtual machine to use for your IDS implementation. The following list enumerates the supported languages and, for interpreted languages, the shebang (the line starting with `#!`) that you should include as the first line of your source file:

C: gcc version 4.4.5

C++: g++ version 4.4.5

JavaScript (node.js 0.6.8): use `#!/usr/bin/node`

Perl 5.10.1: use `#!/usr/bin/perl`

PHP 5.3.3: use `#!/usr/bin/php`

PLT Scheme 4.2.1: use `#!/usr/bin/mzscheme`

Python 2.6.6: use `#!/usr/bin/python`

Ruby 1.8.7: use `#!/usr/bin/ruby`

What to hand in

It is very important that you follow the rules outlined in the Assignments section of the LEARN course site for submitting your assignment. Otherwise we may not be able to mark your assignment and you may lose partial or all marks. By the **a2 deadline**, you are required to hand in:

a2.pdf: A PDF file containing your answers for the written-response questions and optionally your answer to the programming part bonus question.

src.tar: Your source files for your IDS, in your supported language of choice, inside a tarball. To create the tarball, `cd` to the directory containing your code, and run the command

```
tar cvf src.tar .
```

(including the `.`). If you are using an interpreted language, your source should include an executable script named `ids` that runs your code using the proper shebang. For compiled languages, include a Makefile in your source with a default target that will be run inside the virtual machine and builds an executable named `ids`. Do not launch the virtual machine from within your Makefile.

We strongly encourage you to test your IDS in a clean `/share` directory.

TCP/IP Primer

You can skip this section if you've taken a computer networking course. If this section does not answer your questions, Google may know the answer or you can ask on Piazza.

Information on the Internet is transferred using the IP protocol in IP packets. Each IP packet contains in its header a destination IP address (e.g., "129.97.173.41"), a source IP address, and some other fields (see https://en.wikipedia.org/wiki/IPv4#Packet_structure for the detailed packet structure). The addresses identify the packet's destination host and source host, respectively. Routers use the destination IP address to route a packet towards its destination host. This host uses the source IP address of a received IP packet as the destination address of the IP packet(s) that form the host's response. The payload of an IP packet is typically either a TCP segment or a UDP packet.

The TCP protocol runs on top of the IP protocol. It provides reliable, connection-oriented data transfer between a client and a server. An application (such as a Web server) passes a data stream to the TCP protocol, which splits the stream into individual TCP segments. In turn, a TCP segment is put into the payload of an IP packet. A TCP segment has its own header (see https://en.wikipedia.org/wiki/Transmission_Control_Protocol#TCP_segment_structure). This header contains a source port, a destination port, a sequence number, an acknowledgment number, and some flags.

The two ports are used by the TCP protocol to (de)multiplex between multiple servers (or clients) running on the same host. Particular types of servers typically run on pre-assigned ports (e.g., Web servers run on port 80), whereas clients run on a port that is randomly chosen for each connection. To achieve reliability, each TCP segment is given a sequence number by the sending host. The receiving host uses the acknowledgement number in the response packet to acknowledge receiption of a segment. Unacknowledged segments will be retransmitted by the sending host. There are also multiple flags in the TCP header. The SYN flag is used during connection setup. A client that wants to open a connection to a server sends an empty TCP segment whose SYN bit is set to this server. The server acknowledges this TCP segment by creating an empty TCP segment of its own, which also has its SYN bit sent. Finally, the client acknowledges the server's TCP segment in a third TCP segment, and the connection is established (see http://www.inetdaemon.com/tutorials/internet/tcp/3-way_handshake.shtml for details).

The UDP protocol also runs on top of the IP protocol. It is neither reliable nor connection-oriented. Therefore, its header is much simpler (see https://en.wikipedia.org/wiki/User_Datagram_Protocol#Packet_structure). Similar to TCP, the header contains a source port and a destination port.

The most popular application protocol to run on top of UDP is DNS. DNS is used by an application (e.g., a Web browser) to map a domain name (e.g., "www.cbc.ca") to an IP address. In turn, this

IP address is used as the destination address of the IP packets that are sent by the browser to the Web server. DNS servers run on port 53. There are DNS queries and DNS responses. Both of them contain the queried domain name. In addition, a DNS response contains a set of answer records. There are different kinds of such records. In the assignment, we are concerned with records of Type A (i.e., IP addresses). See https://en.wikipedia.org/wiki/Domain_Name_System#DNS_message_format.

In the sample input handed out, you may also come across some other packets. For example, you may see ARP packets. ARP packets operate at the intersection between the IP layer and the link layer, which is below the IP layer. ARP packets are used in a local network to find the MAC address of a node with a particular IP address. Whereas IP addresses are used to route packets globally across the Internet, MAC addresses are used to deliver a packet to a specific node in the local network. There are ARP requests and ARP replies.

Tcpdump displays the contents of each captured packet. Packets are captured at the link layer. The first one or two lines that are output for each packet present some information about the packet in a format that is more readable than hexadecimal format. For an IP packet, the first line lists the timestamp of the packet capture and some fields of the IP header. The second line starts with the following pattern: $a.b.c.d.e > f.g.h.i.j$ where $a.b.c.d$ is the source IP address, e is the source port, $f.g.h.i$ is the destination IP adddress, and j is the destination port. The remaining entries on this line list the other fields in the TCP or UDP header. For DNS packets, the query or response is also shown on the second line for better readability. The remaining lines (starting with `0x0000:`) display the entire contents of the link layer packet in hexadecimal format. Note that the first 14 bytes contain the link layer header, so the IP header starts at the 15th byte (typically containing `0x45`). The payload of a TCP/UDP packet follows the IP and TCP/UDP headers.