**CS 5434 Defending Computer Networks**

**Project 3: Web Exploit Scanner**

**Interim Milestone Document**

**Name: Prashanth Basappa Net ID: pb476**

1. **Objective:**

In this project, the task is to develop a HTTP proxy which can interact with a virtual machine and replay contents from a server(web page) and detect if it is malicious or not. This web exploit scanner can detect malicious websites/domains and report it back to the client browser. The HTTP proxy will handle connections to the client (browser), server (web page) and virtual machine via sockets. It keeps track of the web pages parsed in a local buffer (~64KB).

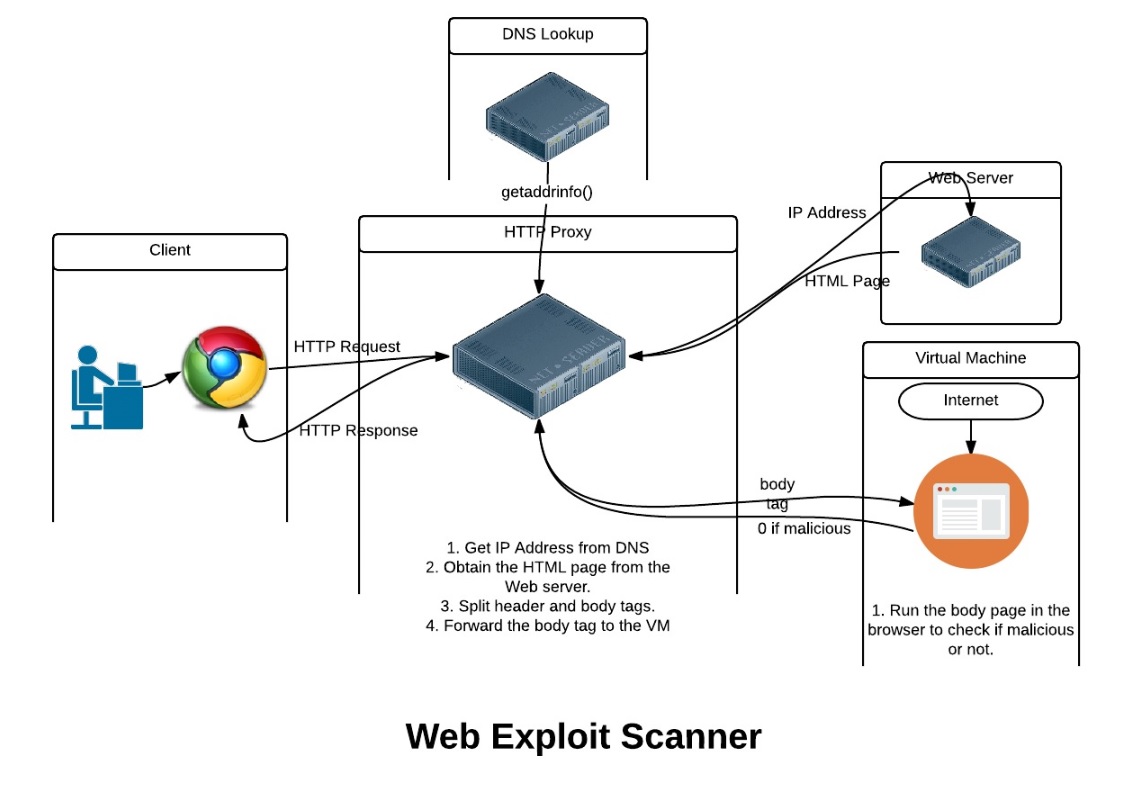
1. **Basic Introduction:**

HTTP is a client-server protocol. The client (usually your web browser) communicates directly with the server (the web server software). However, in some circumstances it may be useful to introduce an intermediate entity called a proxy. Conceptually, the proxy sits between the client and the server. In the simplest case, instead of sending requests directly to the server the client sends all its requests to the proxy. The proxy then opens a connection to the server, and passes on the client's request. The proxy receives the reply from the server, and then sends that reply back to the client. The proxy is essentially acting like both a HTTP client (to the remote server) and a HTTP server (to the initial client).

Why use a proxy? There are a few possible reasons:

* Performance: By saving a copy of the pages that it fetches, a proxy can reduce the need to create connections to remote servers. This can reduce the overall delay involved in retrieving a page, particularly if a server is remote or under heavy load.
* Content Filtering and Transformation: While in the simplest case the proxy merely fetches a resource without inspecting it, there is nothing that says that a proxy is limited to blindly fetching and serving files. The proxy can inspect the requested URL and selectively block access to certain domains, reformat web pages (for instances, by stripping out images to make a page easier to display on a handheld or other limited-resource client), or perform other transformations and filtering.
* Privacy: Normally, web servers log all incoming requests for resources. This information typically includes at least the IP address of the client, the browser or other client program that they are using (called the User-Agent), the date and time, and the requested file. If a client does not wish to have this personally identifiable information recorded, routing HTTP requests through a proxy is one solution. All requests coming from clients using the same proxy appear to come from the IP address and User-Agent of the proxy itself, rather than the individual clients. If a number of clients use the same proxy (say, an entire business or university), it becomes much harder to link a particular HTTP transaction to a single computer or individual.

1. **Design flow of the project:**



**3.1 Modules involved:**

1. Client:

The client is involved as a user by using the browser. The web exploit scanner has the job of alerting the client if the requested website is malicious or not. For testing, both malicious and safe web domains are passed to test the scanner. The client is connected to the HTTP proxy by establishing a socket connection.

1. DNS Resolver:

The DNS (Domain name system) resolver accepts the hostname and resolves it into its corresponding IP address and sends it back to the HTTP proxy. This can be done using the getaddrinfo() command from the sys/socket library.

1. HTTP Proxy:

When the proxy starts, the first thing that it will need to do is establish a socket connection with the client and that it can use to listen for incoming connections (HTTP requests). Your proxy should listen on the port specified from the client browser or the command line and wait for incoming client connections. Each new client request is accepted, and a new process is spawned using fork() to handle the request. Once a client has connected, the proxy should read data from the client and then check for a properly-formatted HTTP request. The request is parsed and the hostname is retrieved and sent to the DNS resolver. The IP address is obtained using the getaddrinfo() command. It is also connected to the server of the web page from where it retrieves the HTML page of the IP. Then, it is parsed and only the body page is sent to the VM through a socket. When it receives a response from the VM about whether a page is malicious or not, it will report it to the client.

1. Web Server:

The server here refers to the web servers of websites we are trying to access to. It will return the web page for a particular IP.

1. Virtual Machine

The VM is running a server and is connected to the HTTP proxy via sockets. It acts as a third party server that is actually checking if a website is malicious or not. It’s responses are recorded in the proxy.

**3.2 Possible flow in the projected design of the project:**

**Step1**: The proxy should establish a socket connection that it can use to listen for incoming connections with the help of sockets. The proxy should listen on the port specified from the command line and wait for incoming client connections. Each new client request is accepted, and a new process is spawned using fork() to handle the request.

**Step 2**: The proxy can inspect the requested URL and selectively access domain names in the HTTP GET request page by parsing it. Then the proxy should read the data and then check for a properly-formatted HTTP request. We can retrieve the IP address of the acquired domain name using the getaddrinfo() command.

**Step 3**: Create our request and set up a socket for talking to the server (web page). Send the request to the server using the bind and accept methods for the socket.

**Step 4**: Receive response from the server and parse the response. Separate the header and body contents after parsing it. For parsing the response contents, I will be using the “\r\n” delimiter which separates the header from the body fields in the response web page.

**Step 5**: Save the data in a small buffer (~64KB)

**Step 6**: Send the body field to the VM to check if the web page is malicious or not.

**Step 7**: Perform the malicious check in the VM and return 0 if malicious and 1 if content is not malicious.

**Step 8**: Save the response from the VM in a log file which may act as a temporary cache for future references from the client. We can use this log file to directly check if the webpage has already been parsed and checked if it was malicious or not.

**Step 9**: The response from the virtual machine is recorded in the proxy and sent to the client’s browser.

1. **Potential Barriers:**
2. *Link Rewriting*: Most dynamic websites include variables in their URLs that tell the site what information to show the user. Sometimes, this kind of URL structure will be transformed into a more human readable form by search engines. It’s an easy URL to remember and to pass it on to another person. But, unfortunately the human readable URL cannot be easily understood by a server. So, a possible obstacle might be to translate the URL to a server compatible mode.
3. Spawning the browser from the local machine.
4. Daemon’s process in VM, forking it and then later on killing it.
5. Malicious detection algorithm.
6. **Potential Enhancements:**
7. Maintaining a log file in the HTTP proxy(temporary cache)
8. Using the execve() command to spawn the browser in the VM from the local machine
9. **How to detect if a website is malicious or not?**

Untrusted input can come from, but is not limited to,

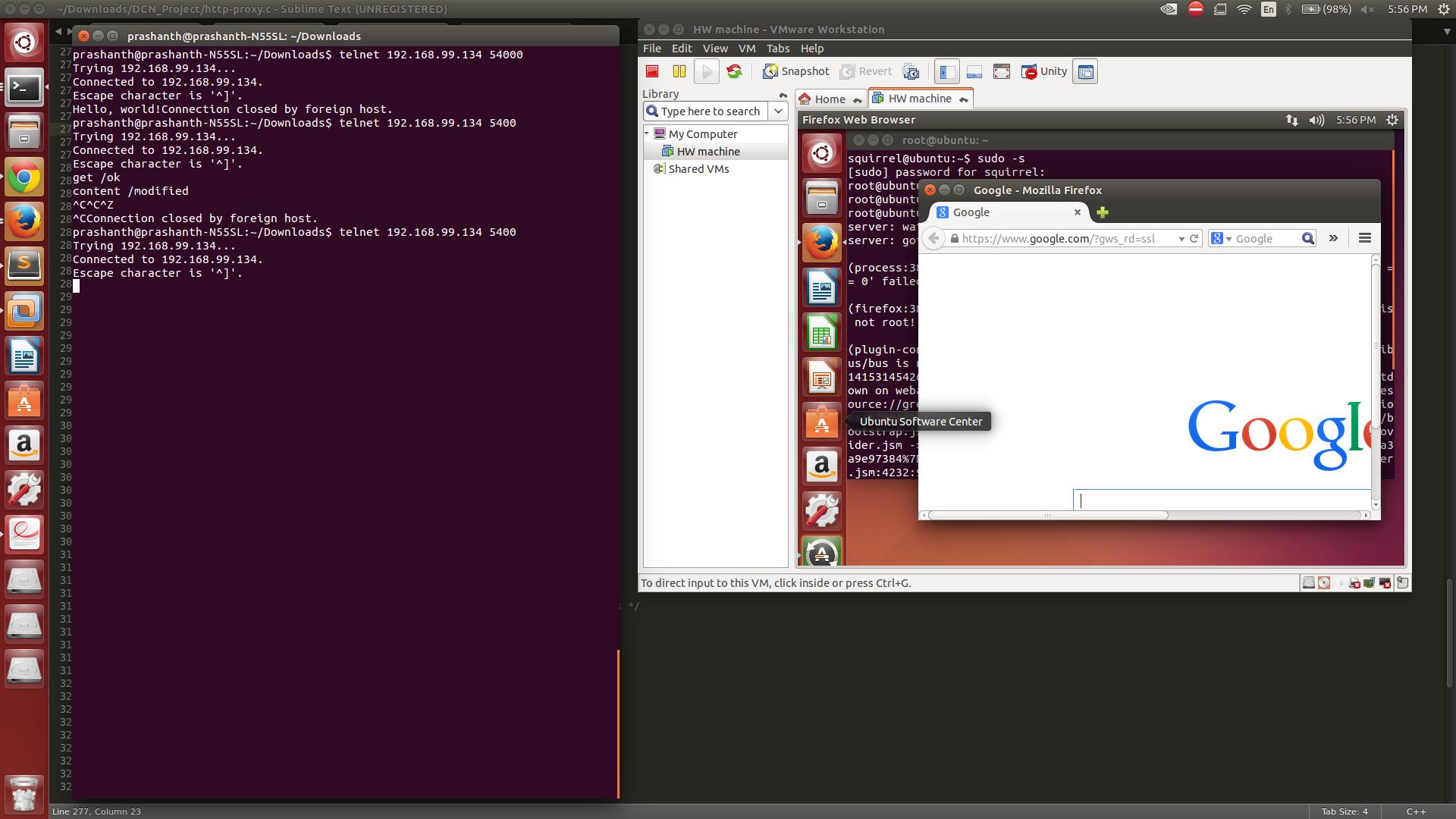
* URL parameters
* Form elements
* Cookies
* Databases queries

My approach would be to check the processes list running on the virtual machine and see if there is any suspicious activity happening. Sometimes, there might be a memory explosion or a creation of multiple child web pages from the malicious web site or there might be sudden increase in RAM usage in the virtual machine. All these factors contribute towards the fact that the website is malicious. I would check the top 10 processes and see if any untoward incident has occurred. The VM will return 0 to the HTTP proxy if it is malicious else 1 if it is client safe. The risk of a web server not doing a check for special characters in dynamically generated web pages is that in some cases an attacker can choose the data that the web server inserts into the generated page. Then the attacker can trick the user's browser into running a program of the attacker's choice. This program will execute in the browser's security context for communicating with the legitimate web server, not the browser's security context for communicating with the attacker. Thus, the program will execute in an inappropriate security context with inappropriate privileges.

An alternate approach would be to check the registers and its contents to see if any vulnerability exists. Also, checking if there is any exploit using heap spaying would be another approach.

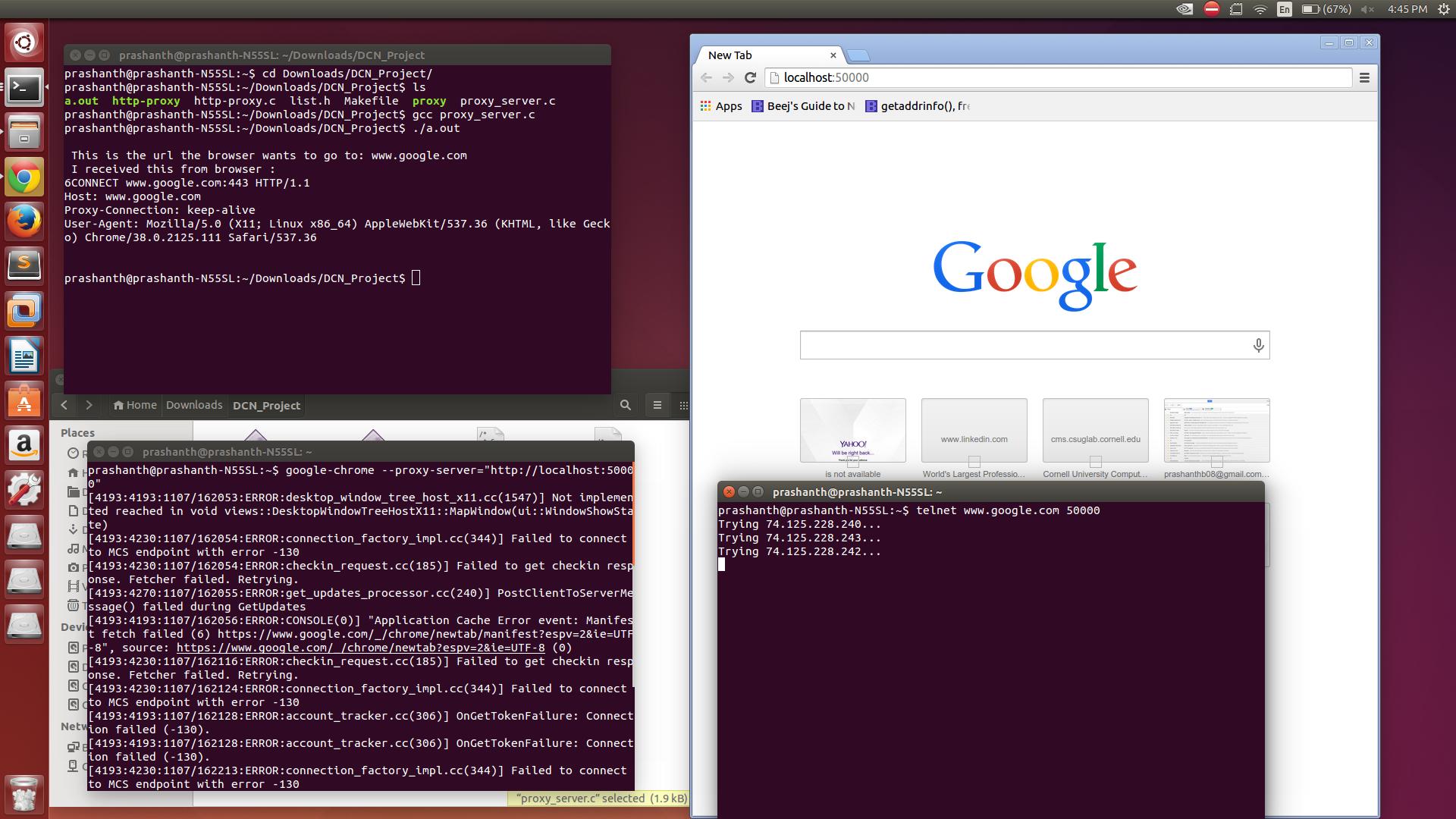
1. **Screenshots taken:**
2. Basic connection setup from the local machine with the virtual machine

This screenshot was taken after a browser was spawned inside the virtual machine from the local machine. I was able to telnet into the VM with its IP address on a specific port(in this case 5400) while the server was running inside the VM. When the server got a response from a foreign host it was able to spawn a browser via the execve() command.



1. Displaying the HTTP GET request in the terminal from the browser

Here the proxy server is started and when the hostname is entered using telnet, its contents is displayed in the server using the read and print command.



1. Basic client-proxy-server(web page) connection:

The two screen shots are taken to show that the basic client-proxy-server relation is established. This is accomplished by running the http proxy from the client’s browser. The proxy gets the request and sends another request to the LinkedIn and Google’s web server. This is established on a separate port (say 50000) and sent back to the client via the original port. I am also able to retrieve the html page of that particular webpage in the terminal using the telnet command to the original port on the localhost.

