CSci 5271: Introduction to Computer Security

Hands-on Assignment 2 **due: Friday, November 22nd, 2019**

Ground Rules. You may choose to complete this assignment in a group of up to three people; working in a group is not required, but strongly recommended. If you work in a group, only one group member should submit a solution, listing the names of all the group members. Use Canvas to submit a tarred and gzipped directory containing all the files mentioned in the questions. The due date for the project is 11:59pm Central Time on Friday, November 22nd. You may use any written source you can find to help with this assignment, on paper or the Internet, but you **must** explicitly reference any sources other than the lecture notes, assigned readings, and course staff.

net-Workin' it This homework includes attacking the network connection between two machines. Because it would be a violation of campus policy to let you actually sniff traffic on a network with other users, we've implemented these machines and the network between them virtually. There is an "attacker" VM that you control, which is similar to the VM you used for HA1. Then on the same network there are a pair of "victim" machines: a "server" VM which hosts a web site, and a "client" VM that simulates an innocent user of that site. You won't be able to log in to the victim machines, but you'll be able to (attempt to) access pages on the server. In a realistic attack you might need to wait for an innocent user to use the vulnerable web site, but to save you time we've set up the simulated client VM so you can poke it via a web request to cause it to perform an access.

Since the virtual machines are on a private virtual network they don't have DNS names and you'll need to refer to them directly by IP address. Each group will be assigned a group number G. Then the IP address of the server will be 192.168.G.1, the simulated client will be 192.168.G.2, and your attacker VM will be 192.168.G.3. There will be some more logistical details about the VM setup posted separately on the course web site. In the directions below we'll use the variables `SERVER` and `CLIENT` to refer to the server and client IP addresses. You'll need to substitute the correct addresses for your configuration, perhaps using shell variables as a short-hand.

The server has some web pages with **extremely valuable** content, which is protected by a variety of techniques. Your mission for this assignment will be to gain access to this content by exploiting weaknesses in each of those techniques. You can start by looking around the web sites yourself, though there isn't too much to see. Your attacker VM has the text-mode web browser `w3m` installed, but you'll probably instead want to use a real graphical browser. The attacker VM isn't powerful enough to run a browser like Firefox, so instead you should use SSH's port forwarding to connect a browser on a real machine to the virtual network. There will be more detailed directions on this on the web site.

1. [10 points]

The web page `http://SERVER/secret/file` is password protected using HTTP "Basic" password authentication. Your `CLIENT` VM knows the password and you can ask it to load the page any time you want with the command "`curl http://CLIENT/1`" (note: no slash after the `1`). Your job is to use `tcpdump` on your attacker VM to sniff the password and

then access the web page. You can learn more about `tcpdump` by typing `man tcpdump` (the manpage exists on the CSE Labs machines, though you aren't allowed to run the program there). To complete this portion of the assignment, hand in two files. One called `dump1.txt` should show the command line you used to run tcpdump, the captured packet with the password, and your guess of the plaintext username and password. One called `file1.txt` should show the contents of the secret file.

2. [15 points]

The web page `http://SERVER/secret/cheese` is protected using HTTP Digest authentication. Your `CLIENT` VM knows this password as well and you can ask it to load the page any time you want by doing `curl http://CLIENT/2` . You should once again use `tcpdump` to sniff the authorization packet. However, this time the packet won't include the plaintext password, but will instead include some mangled bits that are the result of cryptographically hashing the password and some other information. Your job is to show that this is ineffective, by recovering the password with an *offline dictionary attack*. You should implement a short script in your favorite programming language to try hashing various words to see which one is the password. (You probably want to use a language that has an implementation of the MD5 hash function already available; C/C++ with OpenSSL, Java, Perl, Python, or Ruby would all be suitable.) The rules for HTTP Digest authentication are in RFC 2617, or you can find summaries of them in many other places. For this part you should hand in:

- A file named `dump2.txt` that includes the tcpdump command line you used and the packet that you used for your attack.
- A file named `crack2` (perhaps with an appropriate extension like `.pl` or `.c`) that is the source file for your password cracker.
- A file named `readme2.txt` that explains briefly what your password cracker does, and your guess about the password.
- A file named `file2.txt` that is the contents of `http://SERVER/secret/cheese`.

Encryption is not enough. The administrators of this server know that to protect their most important information, they should use SSL/TLS. Thus the remaining questions attack the encryption-protected site `https://SERVER/`. Because the administrators of the server were too cheap to buy a real SSL certificate, the server uses a "self-signed" certificate that your browser will probably warn you about. For this assignment you should click through these warnings, for instance in Firefox you'll want to click on an Advanced button and then "Add Exception" and "Confirm Security Exception." In real life this would be a bad sign about the server's security. In particular, using a self-signed certificate makes this server pretty vulnerable to man-in-the-middle attacks, because anyone else could generate a self-signed certificate that looked just as legitimate. It might be tricky to carry out that attack on our shared virtual network, though, so it's not part of the assignment.

3. [15 points]

The more complex web applications on the HTTPS server use a custom login process and cookies for authentication. In this system, when you log in, the server sets a cookie in your

web browser that later tells the server you are authorized to view various pages. This system is very simple, so their are two kinds of users: the mighty "Stephen" account, and everyone else. Play around with the pages for awhile. You should be able to see all of the cookies for your browser by following appropriate menu choices. In recent versions of Firefox, choose Web Developer | Storage Inspector from the menu button, or in the Tools menu from the menu bar.

The first attack on the HTTPS server is to create a cookie that will allow you browse the site as if you are the privileged user "Stephen". In particular, if you set the cookie correctly, you should be able to load the page `https://SERVER/private/admin` and not get an error message. This will require you to do a little reverse engineering of the format of the cookies used by the server. Depending on your browser, it might provide a user interface of editing cookies, or you can use a command-line program like `wget` or `curl` that lets you manually specify cookies.

For the purposes of grading, you should put your cookie in a file named `cookies3.txt` in the traditional `cookies.txt` tab-separated format used by old Netscape browsers (as well as `wget`, the import-export cookies feature of IE, and other programs). You will hand in the file `cookies3.txt` and a file named `readme3.txt` that explains how you made `cookies3.txt`, and gives the contents of the private page you were able to access.

4. [20 points]

A second part of the HTTPS site uses a database to store thoughts about pictures, at `https://SERVER/thought`. There's supposed to be a privacy feature implemented that keeps you from seeing certain other users' private thoughts (this isn't tied in with the authentication system described in the last question though: it will just always show you the thoughts of a user named "aditya"). The code that implements this web page has a SQL injection vulnerability that allows that protection to be subverted. Your job is to make a request to this page that lets you see a thought that a user named "alice" has recorded about the picture named `char-kway-teow.jpg`. Note that this form only supports POST requests, not GET requests, so you can't make your malicious request just using your browser's URL bar: you'll have to use some other mechanism like a command-line program, a script, or a browser extension. For this question you should turn in a file `readme4.txt` which describes how your attack works, including the particular request you make for the attack, and gives Alice's secret thought.

5. [20 points]

A third part of the site that allows user comments (`https://SERVER/comment`) turns out to be vulnerable to a cross-site scripting attack. Anyone can add a comment, and then all the comments will be available to any user based on a sequential comment number. Because this part of the site doesn't do sufficient sanitization, there's a stored XSS problem if a victim user browses a comment placed by a malicious user. Your job is to post a comment that will allow you to retrieve another special cookie held by the victim user. You can have the victim view your attack comment with the command `curl http://CLIENT/5/COMMENTNUM`, where you replace `COMMENTNUM` with the comment number.

Probably your XSS code will cause the victim's browser to reveal its cookie by making

a web request to some machine under your control. You don't need to set up a whole web server on your attacker machine for this: you can specify any port number in a URL, and if you design the attack so that the information is sent in the initial request, you don't actually need to simulate any response (though if you don't at least close the connection, the client's browser may never finish loading the page). So you could write a simple script or use a program like `netcat`. There are lots of resources about XSS attacks available on the web; one that students have found useful in previous years was written by Amit Klein and entitled "Cross Site Scripting Explained".

Your submission for this part should consist of three files:

- A file named `comment5.html` containing the comment you posted to the web site that contains the XSS attack code.
- A file named `cookies5.txt` containing the cookie you collected from the victim.
- A file named `readme5.txt` that explains how your attack works, and includes any other scripts you wrote and explains any other tricks that were necessary.

6. [20 points]

Because of problems of forging cookies like you showed in question 3, the developers of this web site are starting to get the message that their cookies should be authenticated by including a MAC. This is a good idea, but it turns out they had a bad idea in how to design their MAC. The programming language they use has some cryptographic hash functions built in, so they decided to compute the MAC of cookie contents C by choosing a 20-byte secret key K, using SHA-1 for the hash function H, and having the MAC be $H(C \parallel K)$. But because of a misunderstanding, their implementation truncates the input to H to only 20 bytes (which is the same as the output size). So for instance if C is 10 bytes long, the hash works over C plus the first 10 bytes of K.

This MAC function isn't yet used for cookies on the site, but there's a version of the code as a web page you can test at `https://SERVER/mac-cookie` : the page takes a GET parameter `username` as the cookie contents C, and computes the MAC using the algorithm described in the previous paragraph with a particular secret key.

For this question your goal is to use the poor design of this MAC to break it, by recovering the secret key which would let you forge MACs at will. Because the key is 160 bits long, a brute-force attack will not be feasible: please don't DoS-attack the server! But because of the truncation, a much more efficient attack is possible that recovers the secret key character by character. (Hint: think about MACing a 19-character username, and then the 256 possible 20-character usernames that have that 19-character username as a prefix.) It would be a little bit too cumbersome to implement this attack totally manually, though, so we recommend you write a program to do it.

Your submission for this part should consist of two files: the source code for your attack, in a file named `crack6` (perhaps with a suitable suffix such as `.py` or `.java`), and a file named `readme6.txt` that describes your attack and gives the value for the server's secret key.

Submission checklist. Be sure that your submission includes all of the following:

- `dump1.txt` and `file1.txt`
- `dump2.txt`, `crack2`, `readme2.txt`, and `file2.txt`
- `cookies3.txt` and `readme3.txt`
- `readme4.txt`
- `comment5.html`, `cookies5.txt`, and `readme5.txt`
- `crack6` and `readme6.txt`.

Happy hacking!