Project: HACQIT Exercise

Introduction

One of the goals of this class is to teach how to analyze the security of systems. The *penetration test* is a powerful *a posteriori* testing technique that examines not only the design and implementation of a system but also its maintenance and operation—the latter two often being overlooked in *a priori* evaluations (because the system is not yet fully operational, or is not yet in its production environment).

A proper penetration test devises specific goals that the testers are to achieve. Examples of such goals are to acquire *root* access on a UNIX system, to read a specific file, to create or delete a specific file, or to block access to the system for some period of time. The goals vary depending upon the security policy of the site and upon the reasons for the test.

We will use the Flaw Hypothesis Methodology for this study. The project grade does *not* depend upon achieving the stated goals It depends upon the correct implementation of the Flaw Hypothesis Methodology. You are required to keep a notebook to demonstrate that you are using the methodology. If the notes show what you thought of, tried (and how it was tried), the results, and any ideas that spring from the test, you will get a good grade. If the notes do not show this, you will receive a bad grade even if some of the goals of the test are met. In other words, your grade depends upon your use of the methodology.

Background

This class section will analyze a set of systems implementing a controller designed to provide service even when intrusions occur. Although breaking into the systems is of interest, so is merely inhibiting their response. So this should be a fun exercise!

You are required to follow certain rules. If you break the rules, we can (and will) take appropriate action, up to and including removing your access to the Teknowledge systems and filing a complaint with the SJA. For whatever it is worth, we have never had to do this before, and I'd like to keep this class' good record intact. So, your classmates, the teaching assistants, the researchers at Teknowledge and the UC Davis Computer Security Laboratory, and I would appreciate your cooperation.

You are being granted access to a portion of the Teknowledge network at Teknowledge Corporation for the purposes of this course. You will be given an IP address, a user ID, and a password. Use there as your *exclusive* means to access the systems.

- These user IDs and accounts may ***only*** be used for the purposes of this class and not for any other purpose.
- User IDs and passwords ***must not*** be shared with others. They are for your class use and no other use.
- You must not advertise, in any way, your use of these systems to others. They are just more class systems used in your education and they happen to be operated by and located at Teknowledge.
- Federal law explicitly prohibits the placement of, or transfer of, any pornographic or "inappropriate" material in, to, or from these systems.
- The defenses in these computers do not necessarily reflect any particular defenses used in any other particular computers or networks.
- These computers may include experimental defenses, intentionally weakened or artificially strengthened defenses, and ***any*** defense technologies that the defenders see fit to place in them.
- Teknowledge's research group may be recording your uses of these computers for their research. ***Do not*** attempt in any way to circumvent this recording, unduly obfuscate your methods, or pander to them. Just go about your business as if we never told you this ….

You will only be able to reach the Teknowledge systems by using SSH to enter a secure login server (more on this below). Please ***do not*** proxy back X11—use this SSH connection only for terminal sessions and securely copying files. This is necessary because 70 students proxying back X11 will make the connection to UCD collapse.

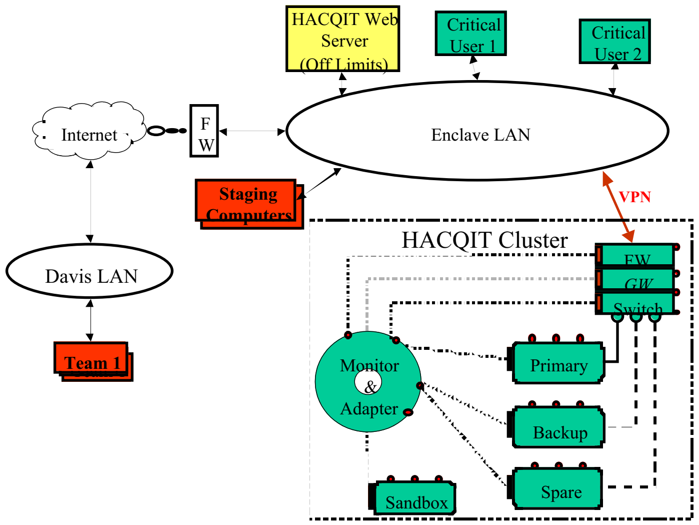
Think of access to the login server as insider access to the network under attack, obtained from an insider who has loaned you her computer so you can break in. The login server is ***not*** to be attacked in any way by your teams because it is a shared resource for your classmates.

The exercises for this class are part of an experiment that Teknowledge and the UC Davis Computer Security Laboratory are running. Your actions on Teknowledge systems will be monitored, and the researchers may use those actions and the results of those actions in their work. Your names will not be used for any purpose except to grant you access to the computers, or to take action if you break the rules.

The HACQIT Systems

The goal of the HACQIT project is to "provide four hours of intrusion tolerance with no more than 25% degradation of aggregate user performance."[1] The project divides servers into two classes: critical and non-critical. HACQIT is concerned only with the critical servers.

The figure below shows the arrangement of the HACQIT network. You will connect to the Teknowledge network



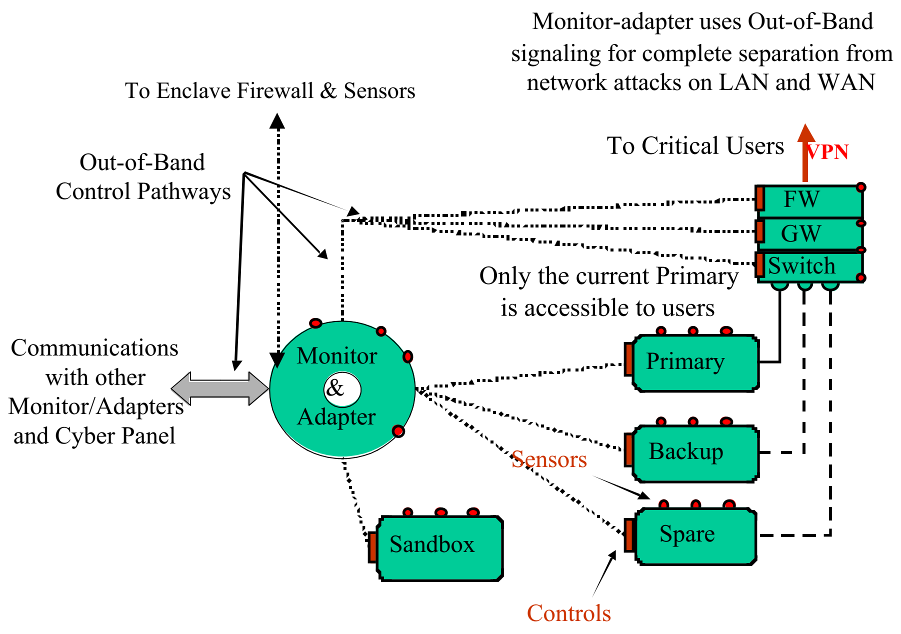
from one of the CSIF systems. This puts you onto a login server ("staging computer" in the diagram above). The targets will be the HACQIT cluster and the paths between that cluster and the critical users. The HACQIT web server connected to the enclave LAN will be used to monitor the attacks and provide mechanisms for you to ask questions of the HACQIT administrators. ***Do not attack that server!*** You get no credit for doing so, and will cause everyone in the class problems and make them *very* unhappy. So please be courteous!

The HACQIT architecture makes several assumptions about the network:

1. The LAN is reliable, cannot be flooded, and is the only means of communication between the users and the servers. This means that the servers will *not* provide the desired level of service should the LAN be flooded.
2. No denial of service attacks are directly launched against critical users.
3. Users and attackers interact with the critical servers over the hardware and software in the LAN and the HACQIT cluster.
4. The HACQIT cluster, its hardware, and software are pristine at start-up and are patched against known vulnerabilities.
5. Critical users are trusted. (*Note*: you are not a critical user!)
6. Unknown vulnerabilities exist.

The HACQIT cluster is illustrated in the figure below. The firewall (FW) controls access to the LAN. It implements a virtual private network (really, an encrypted tunnel) between the critical user and the HACQIT cluster. The primary and backup servers provide the desired services. The monitor and adapter control the servers, and have an out-of-bands communication channel so the services (and defenses) can be reconfigured dynamically as needed. The

1. J. Just *et al.*, "Technology Characterization and Survivability Validation Framework for HACQIT: Hierarchical Adaptive Control for QoS Intrusion Tolerance" (working paper), p. 1.


Monitor-adapter uses Out-of-Band signaling for complete separation from network attacks on LAN and WAN
To Enclave Firewall & Sensors
To Critical Users
VPN
Out-of-Band Control Pathways
FW
GW
Switch
Only the current Primary is accessible to users
Communications with other Monitor/Adapters and Cyber Panel
Monitor & Adapter
Primary
Backup
Sensors
Spare
Sandbox
Controls

sandbox contains a duplicate of the primary and backup servers, and is used to determine whether failures are transient or the result of an attack, and whether a request will cause an error.

The cluster will be shutdown for daily maintenance each morning and, if needed, at other times. Information on LAN and cluster status will be posted on the HACQIT web server on the test enclave LAN, which (again) is ***not to be attacked***.

The critical application is a web-enabled message board running under either Microsoft's Internet Information Services (IIS) or the Apache web server.

The protection mechanisms and software on the HACQIT cluster for the class will likely change over the course of the term. The HACQIT team will notify the class of these changes as they occur.

Finally, the HACQIT system is still under development. It's very far along, and we do not expect problems. But, unexpected problems may occur. If that happens, please notify the HACQIT team at hacqit@teknowledge.com, and bear with them as they fix it.

Goals

There are three possible desired outcomes for this penetration study. You should try to do as many as possible, but expect not to do all. The goals are, in (what we think is the) order of difficulty:

1. Degrade the service provided by the HACQIT cluster to users on the target LAN by more than 25%. You must stay within the assumptions of the HACQIT project. Specifically, you may not launch denial of service attacks directly against critical users or flood the LAN.
2. Gain access to the HACQIT cluster so that you can cause desired changes on it. Desired changes (in increasing difficulty) are:
   a. alter any file (*proof*: identify the altered file by name and its previous and subsequent content along with how you did it);
   b. gain superuser access (*proof*: show the contents of a root-only accessible file);
   c. add an account for yourself (*proof*: show the new password file and provide us with the name, UID, and password of the new account); and
   d. disable or shut down the server (*proof*: some evidence showing the server is not running, and we will verify this with system logs).
3. Gain access to the virtual private network (*VPN*) between the HACQIT cluster and the user on the local LAN. You will have to insert commands or data that are interpreted by either end to demonstrate this, or supply the

cryptographic keys for some specified connection.

When you attack successfully, please leave your team name behind. This will help the observers correlate log information.

Notebooks

In what follows, a "vulnerability" is any method that will achieve any of these goals.

We will discuss the Flaw Hypothesis Methodology in class. It is also covered in the textbook (see section 19.2.4), Your notebook must document your use of this methodology. It should consist of four types of entries.

Knowledge of the System

These entries document things you have learned about the system. For example, if you establish that the server is an SCO system, you would create an entry saying that and explaining how you know it is an SCO system. Your observations may either be used to reach one of the goals, or to provide background information of other entries.

Hypotheses

These entries document suspected vulnerabilities in the system. In addition to the hypothesized vulnerability, you must say why you think the vulnerability might exist (for example, the system is a Linux Red Hat 6.2 system, and the vulnerability is known to exist for that system). You also have to say what the consequence of the vulnerability would be (for example, if this buffer overflow in the setuid to *root* program succeeds, you can execute an arbitrary program and thereby get access to *root*). Indicate which goal (or goals) the hypothesized vulnerability would help you achieve.

Testing

Choose at least 10 hypothesized vulnerabilities and design tests that will tell you if the vulnerability exists and is exploitable. You do not need to exploit the vulnerability for the test (although many times that is the only way to test). You must be able to carry out the test. For example, you cannot say, "the system administrators should check the contents of the protected file /etc/errors, and if the contents of that file begins with 'abracAdabra' the system is vulnerable." If you can't read the file, you'll have to devise some way of testing whether the file contains what you think. One way is to try to exploit the vulnerability … you get the idea.

Document each test in detail. We will need to be able to repeat it. For each test, document the relevant parts of the system and environment settings, the arguments to the program, the input, and the output, and any relevant side effects (like creating s setuid-to-*root* shell).

Generalization

Some of the tests you documented in the previous entries will succeed. Others will fail. From this, you may get ideas for other vulnerabilities, or be able to generalize to find new vulnerabilities. For example, if a buffer overflow allows you to obtain *root* privileges, and the bug is in a library call, check other programs for use of that library call. They are also vulnerable, in all probability. Document any generalizations, and hypothesize new vulnerabilities based upon them.

Staged Exercise

The penetration exercise will proceed in stages, each stage being approximately two weeks. Initially, the system will be fairly open. Successive stages will tighten security. This way, you will be able to become familiar with the system as you proceed through the stages. You will be required to submit your notes at the end of each stage.

The stages are:

Stage 1. Gaining Familiarity with HACQIT

In this stage, you will have full access through the firewall from the login servers on the enclave LAN. For this stage, you should *not* try to penetrate anything. However, you *can* gather information for future use.

1. The goal of this stage is to gain basic familiarity with HACQIT, not to attack the cluster. It will also allow the HACQIT team time to ensure that the student load can be met with the proper number of login servers, *etc*. The HACQIT team will leave the firewall and cluster relatively open so that you can familiarize yourself with its

operation.

2. Each team will have a user account for the designated login server on the test enclave LAN. You are to log into these machines from CSIF lab machines using SSH. The login servers will also be equipped with another VPN product (likely FreeS/Wan) to connect to the cluster firewall. They will also have the *lynx* browser so that you can try out the critical application, a simple web-enabled message board running under ISS or Apache (as described above). Please familiarize themselves with inserting messages.
3. During this stage, you will have open access through the firewall from the designated login servers using the above VPN. Each team will be provided with a VPN name and password for designated login servers so they can access the cluster.
4. Two critical users will be simulated on two other workstations running Microsoft Windows 2000. ***These are off limits to denial of service attacks throughout all stages of this exercise***. These workstations will be hardened and, perhaps, set up as multi-user systems. For the first stage, the personal firewalls on these machines will be opened up. The normal workload of posting messages to the critical application and normal VPN login will be simulated for both critical users.

If you notice that the login server(s) for the class is (are) overloaded, crash, or have other problems, please notify the HACQIT team test administrator at hacqit@teknowledge.com.

Stage 2. Initial Attacks: Only VPN Traffic through Firewalls

This stage allows attacks against a relatively open system. The HACQIT system, enclave LAN, and critical user workstations will have no significant change from stage1. The HACQIT cluster firewall will only allow traffic through the VPN, and the personal firewalls on both critical user workstations will be left open.

Stage 3. More Involved Attacks: Student Access to VPN Removed

This stage allows attacks against a less open system. In this stage, student user accounts on the firewall for the VPN will be removed, thereby removing your access to the VPN. But the critical user workstation firewalls remain open.

Stage 4. Attacking A Fully Operational HACQIT: Critical User Workstation Firewalls Closed

This stage has the HACQIT cluster fully operational. The critical user workstation firewalls will be configured to the settings for a fully operational system. During this stage, additional systems may be added and the configuration changed. Notices of these changes will be posted to the HACQIT web site.

Organization

For this project, you will need to work in teams of 2. You must pick someone in your own section to work with (the other section is doing a different penetration study). Once you have selected your team, please email the team member names and email addresses to cs153@cs.ucdavis.edu. Members of a team may exchange information freely; the project is their joint work. However, different teams may *not* communicate. Each team works independently of all other teams. This is necessary for two reasons. First, if two teams independently come up with the same vulnerability, they can both use it; were the teams communicating, there would be questions of priority that we want to avoid handling. Secondly, your efforts will be monitored for research purposes, and a copy of your notes will be sent to the researchers. Collaboration across teams will bias the results and limit the effectiveness of the research.

Conclusion

The goal of this project is to help teach you ways to analyze systems for security. It will also bring to life some of the concepts we will discuss in class. Also, in order to defend a system, you need to know how to figure out where you might be attacked. In the instructor’s experience, those who know how to attack are much better defenders because they can react, and understand, much more quickly and effectively than can those who have only check lists of things to look for, or who do not know how attackers think.

User Accounts and IP Addresses

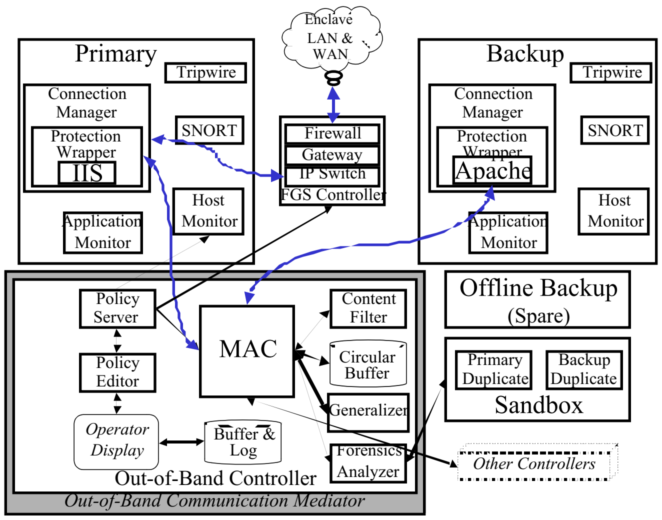
Information about logging into the Teknowledge login server is available on the class web page. Please go to *http://my.ucdavis.edu*, sign in, go to the class web page, and from there to *projects/hacqit/ids.html*.

Important Addresses

HACQIT test administrators' email address: hacqit@teknowledge.com.

Appendix

Just in case you are curious, here is a diagram showing a (simplified) picture of the software implementation of the HACQIT cluster.



The following table summarizes the complexity of the components.

Module	**Language**	**Lines of Code**	**Comment**
Connection Manager	Java	819	
MAC	Java	3589	
Policy	Java	2592	
Policy Editor	Java	3871	
Util	Java	569	
	Sub-total	**11440**	
Application Protection	C++	1930	Wrapper
Application Monitor	C++	773	
WCW	C++	2143	Wrapper / ISAPI Filter
Content Filter Bridge	C++	336	JNI
Forensics	C++	1398	
Monitor lib	C	653	Utility used by HM/AM
Native Bridge	C++	9081	JNI
String Utils	C	1576	
hacqithm	C/C++	6502	Host Monitor
Suspicious Activity Monitor (SAM)	C	406	Checkpoint sample code
	Sub-total	**24798**	
15 modules	**Total**	**36238**	