JANUARY 2011

UILU-ENG-11-2202
CRHC-11-02

ANALYZING SECURITY VULNERABILITIES AND ATTACKS

Long Wang, Nidhi Doshi, Zbigniew T. Kalbarczyk, Ravishankar K. Iyer, Aashish Sharma, James J. Barlow

Coordinated Science Laboratory
1308 West Main Street, Urbana, IL 61801
University of Illinois at Urbana-Champaign

REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 2011	

4. TITLE AND SUBTITLE
Analyzing Security Vulnerabilities and Attacks

5. FUNDING NUMBERS

6. AUTHOR(S)

Long Wang, Nidhi Doshi, Zbigniew T. Kalbarczyk, Ravishankar K. Iyer, Aashish Sharma, James J. Barlow

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 West Main Street
Urbana, Illinois 61801-2307

8. PERFORMING ORGANIZATION REPORT NUMBER
UILU-ENG-11-2202
CRHC- 11-02

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official position, policy, or decision, unless so designated by other documentation

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Analysis of network security attacks helps us understand characteristics of application vulnerabilities, intrusion detections techniques and attacker behavior patterns established. Intrusion detection tools and signature-based approaches used in practice are helpful in detecting known attacks but are not as efficient when a new vulnerability is being exploited. Anomaly-based approaches are sometimes able to detect unknown attacks but at the cost of false alarms. Hence there is the need for human expertise intervention in attack investigation. Today, most of this investigative analysis is done more or less manually. It is our aim to propose a measurement based model to provide joint insight into attack patterns and associated vulnerability exploitation. This paper describes a novel approach that combines both vulnerability data from Mitre CVE (Common Vulnerabilities and Exposures) vulnerability database, and attack data from nearly 5000 hosts at National Center for Supercomputing Application (NCSA) located at Illinois, to create an attack model with sufficient details to assist identification of system compromises. Real data is scrutinized to find attack flow patterns, which give more insight into strategies adopted by the elite attacker community today. Motivated by these findings, we create a model to capture the relationships between vulnerability exploitation and attack flow.

14. SUBJECT TERMS
vulnerability, attack, real data, classification, model.

15. NUMBER OF PAGES
21

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Analyzing Security Vulnerabilities and Attacks

Long Wang, Nidhi Doshi,
Zbigniew T. Kalbarczyk, Ravishankar K. Iyer
Center for Reliable and High-Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana Champaign
{longwang, ndoshi, kalbarcz, rkiyer}illinois@edu

Aashish Sharma, James J. Barlow
National Center for Supercomputing Applications
University of Illinois at Urbana Champaign
{aashish,jbarlow}@ncsa.uiuc.edu

Abstract

Analysis of network security attacks helps us understand characteristics of application vulnerabilities, intrusion detections techniques and attacker behavior patterns established. Intrusion detection tools and signature-based approaches used in practice are helpful in detecting known attacks but are not as efficient when a new vulnerability is being exploited. Anomaly-based approaches are sometimes able to detect unknown attacks but at the cost of false alarms. Hence there is the need for human expertise intervention in attack investigation. Today, most of this investigative analysis is done more or less manually. It is our aim to propose a measurement based model to provide joint insight into attack patterns and associated vulnerability exploitation. This paper describes a novel approach that combines both vulnerability data from Mitre CVE (Common Vulnerabilities and Exposures) vulnerability database, and attack data from nearly 5000 hosts at National Center for Supercomputing Application (NCSA) located at Illinois, to create an attack model with sufficient details to assist identification of system compromises. Real data is scrutinized to find attack flow patters, which give more insight into strategies adopted by the elite attacker community today. Motivated by these findings, we create a model to capture the relationships between vulnerability exploitation and attack flow.

Keyword: vulnerability, attack, real data, classification, model.

1. Introduction

While a plethora of techniques to protect against a myriad of attacks are available, there is relatively little data to quantify attack scenarios, especially in a joint fashion with the associated vulnerabilities. Measurement-based comprehensive assessment, starting with discovery of attacks and ending with evaluation of the compromise severity, has not been well explored.

The current methods used in practice for identifying attacks and attack sources, and compromise assessment, substantially combine online measurements with human expertise. These processes, while somewhat slow and inefficient, offer a unique opportunity for developing a real-life understanding and analysis. National organizations like NCSA (National Center for Supercomputing Applications), which are accessed by large number of users (local, national, and international) and constantly under attack, collect a wide variety of data on attack activity. Due to the fact that they use many different types of machines and applications, this serves as the invaluable source of our study data. Many vendors measure and maintain their own databases of

attacks and provide timely patches for the vulnerabilities detected. Typically, this information is not public and is geared to maintain vendor specific systems. Their analysis is naturally proprietary.

The NCSA has been meticulously working towards setting up their systems to collect intrusion alert, surveillance, and compromise detection information. This information combined with the additional insight provided by the human experts (who perform both online and offline analysis) presents a unique opportunity to conduct a comprehensive study. This data, when combined with publicly available data on known vulnerabilities, allows the overall modeling of an attack from inception to system compromise. The methodology developed provides two levels of insights.

Attack-flow analysis and attack-model generation. The attack measurement and joint attack/vulnerability analysis approach consists of two steps: (a) separate analysis of the attack patterns resulting in a model built with the help of the NCSA security team, (b) analysis of the CVE data to relate the vulnerabilities to the consequent compromises. Combining these two information sources, the event patterns pertinent to an attack are abstracted into an attack model with sufficient detail to assist in identification of system compromises. For example, in many cases, after compromising the machine, an attacker must download and install additional tools to cause actual damage to the system. Consequently, if the attack can be detected during, for example, the abnormal tool downloading stage (e.g., a file downloading without a user-controlled *tty* session), then the attack can be foiled and any potential loss of information or resources can be minimized. In each case, the steps from the start of an attack to the compromise stage are depicted as a finite state machine model.

Extension of the attack-model to a large set of vulnerabilities. The attack models derived from detailed analysis of a relatively small number of real attacks can be extended to depict a large set of vulnerabilities. We make two observations, which enable extending the attack model:

(i) *common/similar consequences of attacks (impacts) exploiting same category of vulnerabilities* – attacks exploiting vulnerabilities of a given category can be mapped into a relatively small set of consequences observed at the system and/or application levels. For example, most buffer-overflow vulnerabilities exploited lead to either a denial of service or an arbitrary binary code execution.

(ii) *common/similar attack-flow patterns for attacks causing the same type of consequences to the victim system* – the post-exploitation machine manipulation by attackers is correlated to the type of impact the attack has on the system and/or application. For example, if an attack (regardless what

vulnerability is exploited) leads to an arbitrary command being executed, the attacker may often issue a well-defined set of commands such as "ps", "uname", "w" without a user-controlled *tty* session. This creates an opportunity for early detection of an intruder.

2. Related Work

Classification and statistical analysis of security vulnerabilities. Aslam et al. [1] use a decision tree for classification of vulnerabilities. Whalen et al. [2] design a grammar to express network protocol exploits and classify protocol vulnerabilities using the grammar. Empirical category classifications, like [5] [13] and the *Bugtraq* classification, are also available and frequently used. Krsul [4] provides a good summary of a number of classification techniques, including an impact-based classification which partitions vulnerabilities according to the attack consequence, or impact, resulting from vulnerability exploitation. Teipenyuk et al [14] also develop an impact-based taxonomy of vulnerabilities. Berghe et al. [3] classify vulnerabilities associated with web service and also conduct a correlation study. There are a number of sources for vulnerability data such as CERT [9], CVE [10][11], and Bugtraq [12]. These sources are widely used for analyzing vulnerabilities in various viewpoints. For example, Bugtraq data are used in [3] [5]. Example vulnerabilities are used for evaluating classification techniques in [4] [13].

Analysis of attack data. In addition to vulnerability classification, attack classification/clustering is also proposed to analyze attacks [6] [8]. For example, Cohen lists 94 types of attacks and 140 types of defenses in [6]. Sung et al. [16] perform a thorough statistical analysis of real traffic data on a single denial-of-service attack on a server at Georgia Institute of Technology. Data mining techniques have been used by several authors to analyze traffic data during attack. This approach is the basis for several intrusion detection schemes, e.g. a rule mining mechanism [17], policy-based analysis in Bro (three levels for specifying/enforcing a policy: packet-event-policy) [18], and statistical methods for atypical behavior extraction in IDES [19]. These approaches apply probabilistic mechanisms for characterizing attack traffic data.

Honeypots have been frequently employed for analyzing real attack data [20]. Cukier et al [7] collect attack data for a two-node honey-net over 109 days, and analyze the data with the goal to find traffic characteristics to separate attacks into different clusters. McGrew et al [21] present an analysis of attack data obtained through a honeypot.

Modeling vulnerabilities/attacks. Using real data from CERT and Bugtraq, Chen et al [5] develop a finite state machine-based model for modeling vulnerability exploitation in the code level. Giffin [22] performs static analysis of the binary code of a program for modeling system calls and data

flows with objective to detect vulnerability exploitation. Ramakrishnan et al [27] develop a vulnerability model in the level of system components, using as a rule-based specification language for vulnerability discovery via formal reasoning.

Ruiu [23] divides the entire process of a typical attack into 7 phases: reconnaissance, vulnerability identification, penetration, control, embedding, data extraction/modification, and attack relay. This phase division is derived from inherent attack logic.

Other formal models of attacks include an attack tree model (with a specification language) [24], a Petri-net model for misuse in attacks [25], and a model capturing protocol attacks involving multiple nodes [26]. They all focus on theoretical study of model properties.

3. Data Sources

Analysis reported in this paper is based on two primary data sources.

Real attack databases are obtained from:

(1) data collected by the on-line instrumentation supported by the NCSA Security team. The data is collected from over nearly 5000 machines including four computer clusters, mail servers, web servers, and other production machines. They run a range of operating systems and processor technologies. The machines are attacked constantly and, based on the last year's data, have approximately one real compromise per week. NCSA has security professionals who have significant experience in identifying/assessing compromises, both on- and off-line. As outlined in Table 1, a range of tools and mechanisms are employed for monitoring runtime environments and providing data for the analysis.

(2) Publicly available data which include security reports provided by security companies/communities, e.g. ISecurityPartner (http://www.isecpartners.com), TippingPoint (http://www.tippingpoint.com) and SecurityFocus (http://www.securityfocus.com).

Figure 1 shows a snippet of Netflow logs for analysis of an *Awstats*[1] attack (irrelevant records are excluded from the figure, indicated as "..." in Figure 1). The snippet shows four actions of an attacker: scan, simple test/environment check, code downloading, and machine use. The first log fragment shows the network scan traffic for establishing a connection from the attacker machine (219.xxx.xx.xx, the real IP address is not disclosed for privacy reasons) to the target machine

[1] AWstats is a free tool for analyzing and reporting data from internet service. It parses and analyzes server log files and produce visual reports on internet service activities.

(141.xxx.xx.xx). During the scan, the attacker (actually the attack script) tries four paths for locating a specific file (*awstats.pl*) to check if the Awstats toolkit is installed on the target machine. After the script determines that the toolkit exists on the machine, the human sends simple commands to test the vulnerability exploitation and check if a user is currently working on the machine (the command "w"). Then the attacker downloads a file *1.tgz* from the machine 81.xxx.xx.xx onto the victim machine through an HTTP *GET* request (the "wget" command is not logged by Netflow, which logs network traffic), as shown in the fragment 3. The last fragment in Figure 1 depicts how the attacker uses the machine. In particular, we see that the attacker removes the file *1.tgz* to hide the footprint of the attack.

Table 1: Data sources for attack analysis in NCSA

Data source	Description
Cisco Netflow [15]	Monitors unidirectional packet streams between a source and destination (<IP, port> pairs).
Argus [28]	Similar to Cisco Netflows, but for bidirectional streams.
Bro [18]	Network-based intrusion detection system. Monitors multiple services including ftp, http, alarm, prof, etc.
UNIX Syslog	Standard UNIX logging. Records activities of users and services.
ModLogSpread	An Apache module for web logging. Collects access and error logs for all of web servers managed, including HTTP request and response headers.
Tripwire [29]	A toolkit for reporting file system anomalies.
Dialup Log	Logs dialup connections, including call ID, user ID, source phone number, etc.
DHCP Log	Logs DHCP requests.
ARP table	Records the mapping table for address resolution protocol (ARP).
LaBrea [30]	Takes over unused IP addresses on a network and creates "virtual machines" that answer to connection attempts.

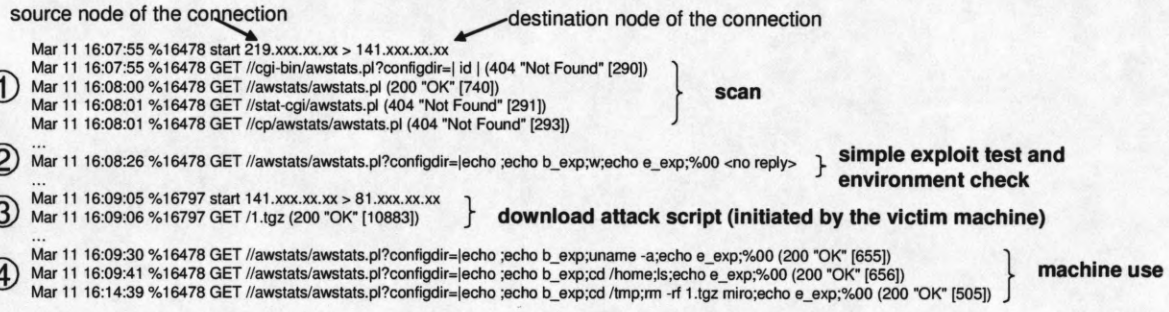


Figure 1: Snippet of real data in attack database

The Vulnerability database employed in our study is Common Vulnerabilities and Exposures (CVE) [11]. This is an indexed dictionary that contains references to data across separate vulnerability databases, which are provided by different sources including the security community (e.g. BUGTRAQ [12]), government-funded organizations (e.g. CERT [9]), and enterprises (e.g. IBM, HP, Microsoft, Redhat). As of November 1, 2006, the CVE contained 20,074 unique security vulnerabilities. Although the abundant information provided in the CVE database enables in-depth

and multi-aspect studies of vulnerabilities, the large number of these and other relevant details make analysis of the database a challenge.

①	CVE-2006-1293	Cross-site scripting (XSS) vulnerability in index.php in Contrexx CMS 1.0.8 and earlier allows remote attackers to inject arbitrary web script or HTML via the query string (PHP_SELF).
②	CVE-2006-1294	PHP remote file include vulnerability in PageController.php in KnowledgebasePublisher 1.2 allows remote attackers to include and execute arbitrary PHP code via a URL in the dir parameter.
③	CVE-2006-1295	Cross-site scripting (XSS) vulnerability in recherche.php3 in SPIP 1.8.2-g allows remote attackers to inject arbitrary web script or HTML via the recherche parameter.
④	CVE-2006-1296	Untrusted search path vulnerability in Beagle 0.2.2.1 might allow local users to gain privileges via a malicious beagle-info program in the current working directory, or possibly directories specified in the PATH.
⑤	CVE-2006-1297	Unspecified vulnerability in Veritas Backup Exec for Windows Server Remote Agent 9.1 through 10.1, for Netware Servers and Remote Agent 9.1 and 9.2, and Remote Agent for Linux Servers 10.0 and 10.1 allow attackers to cause a denial of service (application crash or unavailability) due to "memory errors."
⑥	CVE-2006-1298	Format string vulnerability in the Job Engine service (bengine.exe) in the Media Server in Veritas Backup Exec 10d (10.1) for Windows Servers rev. 5629, Backup Exec 10.0 for Windows Servers rev. 5520, Backup Exec 10.0 for Windows Servers rev. 5484, and Backup Exec 9.1 for Windows Servers rev. 4691, when the job log mode is Full Detailed (aka Full Details), allows remote authenticated users to cause a denial of service and possibly execute arbitrary code via a crafted filename on a machine that is backed up by Backup Exec.
⑦	CVE-2006-1300	Microsoft .NET framework 2.0 (ASP.NET) in Microsoft Windows 2000 SP4, XP SP1 and SP2, and Server 2003 up to SP1 allows remote attackers to bypass access restrictions via unspecified "URL paths" that can access Application Folder objects "explicitly by name."
⑧	CVE-2006-1301	Microsoft Excel 2000 through 2004 allows user-assisted attackers to execute arbitrary code via a .xls file with a crafted SELECTION record that triggers memory corruption, a different vulnerability than CVE-2006-1302.
⑨	CVE-2006-1302	Buffer overflow in Microsoft Excel 2000 through 2003 allows user-assisted attackers to execute arbitrary code via a .xls file with certain crafted fields in a SELECTION record, which triggers memory corruption, aka "Malformed SELECTION record Vulnerability."

Figure 2: Snippet of real data in vulnerability database

As CVE aims to provide a dictionary of security vulnerabilities, the CVE committee makes efforts to merge duplicate reports of the same vulnerability from multiple sources and convert vulnerability description into a uniform format. The database interface makes it easy to search the database using keywords. We extracted descriptions of vulnerabilities from separate web pages and aggregated them into a file. Figure 2 lists a fragment of this file, where index and description of vulnerabilities are listed. The description contains information of target applications, vulnerability type, direct impact of exploiting the vulnerability, and how the exploitation is performed in attacks.

In Figure 2 information marked by rectangles is used for classifying the vulnerabilities, e.g., 1 and 3 are *cross-site scripting* vulnerabilities. The underlined text, in Figure 2, indicates information used for classifying impacts of exploiting the vulnerabilities, e.g., vulnerabilities marked as 1, 2, 3 lead to arbitrary execution of scripts/HTML.

4. Attack Analysis

This section presents two case studies of real attacks to illustrate the patterns corresponding to the attack.

4.1 PHP Horde Attack

The PHP Horde[2] attack exploits an input-parameter-validation vulnerability (incident CVE-2006-1491 in the CVE vulnerability database): the help viewer of the Horde application framework 3.0 and 3.1, invoked by "Horde/services/help/", receives a parameter "module" and delivers the parameter to the eval() call without validating the syntax of the parameter. Arbitrary PHP script code can be injected and executed by exploiting the vulnerability.

Figure 3 depicts the timeline of the PHP Horde attack for a single victim machine, and Figure 4 illustrates the request/response communication during the attack. The detailed steps are summarized in Figure 5.

In this case the attacker did not explicitly remove the downloaded malicious code *asd.tgz*. However, as the file was downloaded into */tmp*, which is emptied whenever the machine is rebooted/shutdown, the attack footprint is removed after the machine reboots.

In this attack case, the impact resulting from the vulnerability exploitation is arbitrary execution of PHP script code. Our conjecture is that any attack which results in arbitrary execution of PHP code is likely to follow the same pattern of events/steps as identified in this example.

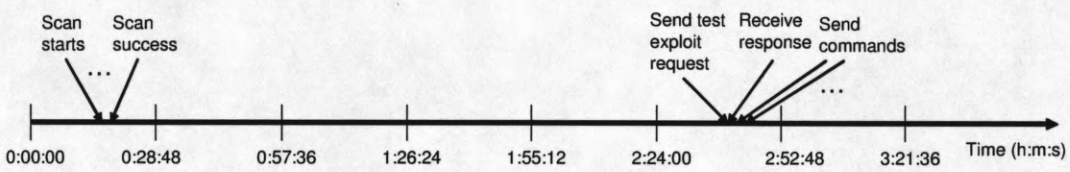


Figure 3: The timeline of the PHP Horde attack for a single victim machine

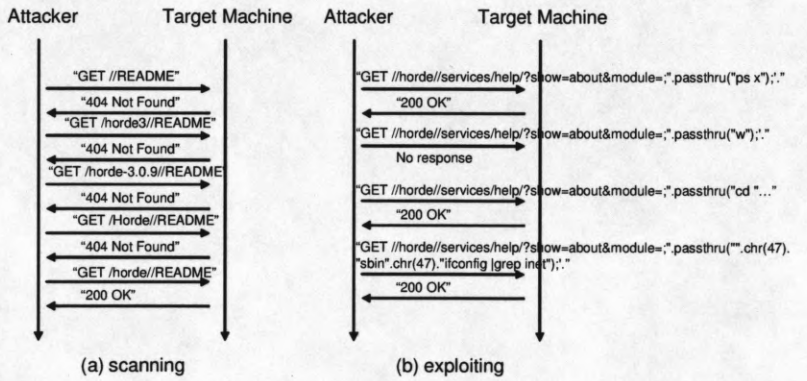


Figure 4: Scanning and vulnerability exploiting in the PHP Horde attack for a single machine

[2] Horde is a web application framework widely used by webmail programs (e.g. IMP), calendar utilities (e.g. Kronolith), address book management utilities (e.g. Turba), and many other web applications (e.g. Mnemo, a group-sharing notepad application).

1. An attack script scans the victim computer by sending varied HTTP *GET* requests (*GET* is a standard HTTP method for requesting a web file and is part of the HTTP protocol) to check existence of the vulnerable Horde application framework, as shown in Figure 4 (a) (the scan for the victim machine starts at 00:16:21, Sep 9);
2. The attack script receives positive response from the victim computer (the scan succeeds for this machine) (00:17:00);
3. The attack script records the success and the victim machine IP address, and continues scanning other machines and networks;
4. Some time later (presumably when the scan of other machines/networks finishes), the attacker reads the record, and sends a request to the victim machine to test if the Horde application framework has not been patched and the vulnerability still can be exploited (02:35:33)

Since the vulnerability allows executing an arbitrary command through PHP code, the attacker first tries two simple tests by sending the following commands to the victim machine via HTTP requests (the command sent is enclosed in the *passthru()* in Figure 4 (b));

"ps x", "w"

The attacker's goals are:

(i) To test if the vulnerability can be exploited. A simple command is usually used for such a test because the success of a complicated command may require knowledge about the specific system environment, and its failure does not necessarily mean the exploitation is futile.

(ii) To get familiar with the target machine environment. Typically an attacker, who comes to a new environment, first tries to get familiar with it before doing anything really meaningful. Here the environment includes both system hardware/software and human activity on the system. In this example, the attacker first tries to see what services are running on the system ("ps x" lists all applications without controlling *ttys*, i.e. user terminals), and then to see whether there are users currently working on the machine ("w" shows who is logged on and what they are doing).

5. From these two tests the attacker confirms (1) the vulnerability could be exploited, and (2) nobody is logged to the machine ("*no response*" in Figure 4(b)), in order to actually exploit the system. The attacker constructs a new request which carries a sequence of commands as given below (the third request in Figure 4 (b), where part of the request is shown), and sends the request to the victim machine (02:41:59):

"cd /tmp ;wget gabryel.lydo.org/asd.tgz ;tar zxfv asd.tgz ;cd .asd ;cp linux /var/tmp/sshd ;/var/tmp/sshd ; /var/tmp/sshd "

The command sequence issued by the intruder performs tasks as follows (i) download a malicious code (*asd.tgz*) into a writable directory (*/tmp*), (ii) unbundle the code, and (iii) deploy the code into a writable directory (*/var/tmp*) and execute the code (note that the code is renamed to *sshd* to hide itself). In this example, the operations actually executed by the downloaded code are still unknown. After starting the execution of the malicious code, the attacker issued a command "*/sbin/ifconfig |grep inet*" to get the IP information of the victim machine. The purpose of this command is also elusive to us.

Figure 5: steps of the PHP Horde Attack

4.2 A Phishing Attack

Phishing attacks are becoming more severe with the growing popularity of web applications. This real case of phishing attack is disclosed by ISecPartners company [31]. The attack procedure is listed in Figure 6.

In this attack a security vulnerability of the browser is exploited: the browser submits the broker-site cookie with forms created in a page from another website. Though psychology-based cheating is employed in the real attack case, there are lots of security vulnerabilities that enable injecting arbitrary web scripts or HTML onto a server page, especially for the web 2.0 applications (blog, wiki, personal space, etc.). For example, multiple cross-site scripting vulnerabilities in sBLOG 0.7.1 (Beta) and earlier versions allow remote attackers to inject arbitrary web script or HTML via the *p* and *keyword* parameters in the home page and the search page (CVE-2006-0101). After exploiting this vulnerability, the attacker is able to execute arbitrary web script/HTML, and of course, is able to follow the attack steps listed in this section.

1. The attacker set up a webpage "cybervillians.com/news.html" that consists of an article on stock analysis as well as malicious web scripts and HTML;
2. The attacker posted a message on a stock message board at finance.yahoo.com, and the message recommends a good article on stock analysis and points to the webpage crafted in step 1;
3. The victim, monitoring his stock positions on a broker website, reads the message on the stock message board at finance.yahoo.com;
4. The victim clicks the link in the message. The crafted webpage is loaded and the victim begins to read the article;
5. When the webpage is loaded, the malicious web scripts and HTML code create 5 hidden web frames, each frame populated with HTML forms;
6. The HTML forms are filled with values to perform the following tasks:
(1) turn off email notification which is issued by the broker website when an account transaction or modification is performed;
(2) add a new checking account (i.e. the attacker's bank account) to the victim's stock account;
(3) transfer $5000 from the victim's stock account into the new checking account;
(4) delete the checking account from the victim's stock account;
(5) turn on email notification
Values in the HTML forms are guessed according to the attacker's own experience of using the broker website;
7. A Javascript controls the submissions of these forms in order, one after another with a delay of one second between consecutive submissions. The forms are submitted to the broker website with the authentication cookie as the victim is still in an active session on the broker website;
8. A few minutes later, all the transactions are completed. Then the victim finishes reading the article and leaves the crafted webpage.

Figure 6: steps of the Phishing Attack

Due to space limitation, in this section we present two example attacks selected from a larger number of real attack cases obtained from NCSA data. Overarching observation is that: *common/similar attack-flow patterns are taken for attacks causing the same type of consequences to the victim system.*

5. Categorization of vulnerabilities and impacts (attack consequences)

This section studies the classification of vulnerabilities and impacts to establish mapping or correlation between different classes of vulnerabilities and expected impact of attacks which exploit the vulnerabilities.

5.1 Classification of Vulnerabilities

The total of 19,097 vulnerabilities reported in the CVE database between 1999 and August, 2006 are studied. As proportions of different categories of vulnerabilities change with time, this paper presents the analysis result for the 4,497 vulnerabilities between January and August, 2006. We classify these vulnerabilities into nine categories listed in Table 2.

Other studies also report on vulnerability classification, e.g. [4][5]. Our breakdown is different in two aspects: (1) it differentiates the vulnerability categories by the context of program execution, and (ii) it tries to correlate the vulnerability to potential consequence of an attack. For example, though three of the vulnerability categories listed in Table 2, *SQL injection*, *cross-site scripting*, and *insufficient parameter input validation*, are due to insufficient check of parameter value/syntax, the types of unchecked parameters are different. For the three categories, the involved parameters are SQL command parameter (for *SQL injection*), web script/HTML parameter for cross-site attack (for

cross-site scripting), and packet payload field /file format/data structure field/HTML parameter for service attack (for *insufficient parameter input validation*).

Table 2: Classification of vulnerabilities

Vulnerability category	Description
Buffer overflow	Memory beyond a buffer boundary is filled with input data.
Integer overflow	Integer representation in computer is overflowed during integer arithmetic operations.
Format string	Use of unfiltered user input as the format string in certain C functions that perform formatting (like printf()).
Race condition	Synchronization is not ensured in cases that different execution orders of threads bring inconsistent results.
SQL injection	SQL commands or invalid SQL parameters are injected into an SQL command for execution
Cross-site scripting	A category of vulnerability, typically found in web applications, that allows malicious users to inject code into web pages viewed by other users.
Insufficient parameter input validation	Parameter values provided by user input are not sufficiently validated.
Design/config error	Application specific design errors and configuration errors, as well as implementation bugs.
Unknown	The vulnerability is not clearly understood.

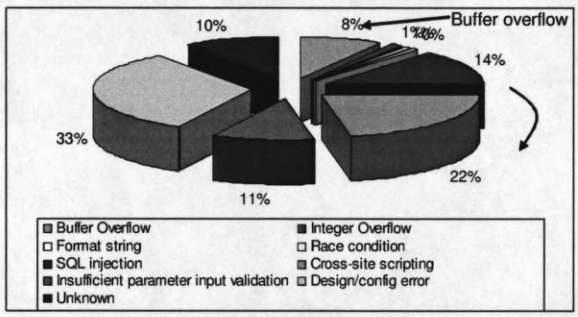


Figure 7: Classification of 4497 vulnerabilities in CVE database (Jan. 2006 – Aug. 2006)

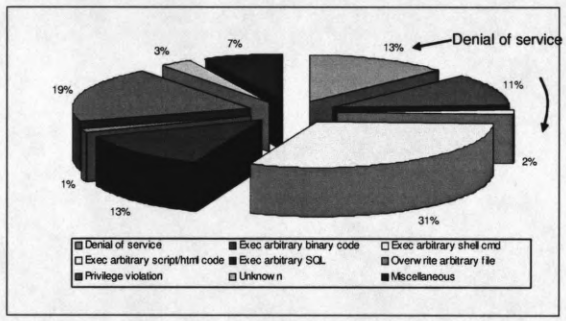


Figure 8: Classification of impacts due to exploiting vulnerabilities in Figure 7

Table 3: Classification of impacts caused by vulnerability exploitation

Impact category	Description
Denial of service	The system crashes/hangs, or the service performance is largely degraded.
Exec arbitrary binary code	Arbitrary instruction code can be injected into the system for execution.
Exec arbitrary shell cmd	Arbitrary shell commands can be injected into the system for execution.
Exec arbitrary web script/ html	Arbitrary web script/HTML code can be injected into the system for execution.
Exec arbitrary SQL cmd	Arbitrary SQL commands can be injected into the system for execution.
Overwrite arbitrary file	Arbitrary files can be created/ overwritten with some data content (but the content can not be arbitrarily provided by the attacker).
Privilege violation	Operations can be performed with privilege check/enforcement bypassed. Includes information leaking, unauthorized access, privilege abuse,
Unknown	The impact of vulnerability exploitation is unclear.
Miscellaneous	Impacts specific to applications and unable to be covered by the categories above.

Most of the categories listed in Table 2 are self-explanatory. The d*esign/config error* is a broad category which covers all vulnerabilities due to errors of application/system design, implementation, deployment, and configuration. Examples include missing of privilege check, implementation bugs,

unexpected runtime/workload scenarios, default/trivial user accounts and passwords, type-unmatched/malformed data, failure of implementation in enforcing a predefined security goal.

Figure 7 shows the classification of the vulnerabilities reported to CVE database for the studied period. According to the figure, most of vulnerabilities fall in categories of *design/config error* (33%), *cross-site scripting* (22%), *SQL injection* (14%), and *insufficient parameter input validation* (11%, most of them are related to web service). Note vulnerabilities in these categories are high due to the popularity of web services and script/HTML code execution, compared with the classification given in [5], where boundary condition error (i.e. buffer overflow) is the largest category.

5.2 Classification of Impacts

Impacts of vulnerability exploitation are classified into nine categories, as shown in Table 3. Different impact categories can be associated with different attack flow patterns and different kinds of damages/losses. For example, a *denial of service* attack usually leads to system crash/hang or significant degradation of performance, which makes the system unusable. In contrast, an *execute arbitrary shell command* attack allows the attacker to interact (by issuing commands) with the victim machine.

Figure 8 depicts the classification of the impacts of exploiting different vulnerabilities shown in Figure 7. Note that sometimes a vulnerability can cause multiple types of impact. Therefore, the number of impacts (4789) is slightly larger than the number of vulnerabilities (4497).

Data shows that the most frequent impact resulting from vulnerability exploitation fall in to four categories: *execution of arbitrary script/HTML* (31%), *privilege violation* (19%), *execution of arbitrary SQL command* (13%), and *denial of service* (13%). With a *privilege violation*, the attacker is able to read/write sensitive information, or obtain a privilege not granted to him/her. The *miscellaneous* category (7%) of impact consists of multiple categories with a small number of cases in each. Examples include: file hiding, man-in-the-middle attack, easy-to-guess passwords, file creation with incorrect permission, preventing security detection tools from working correctly, connecting to an arbitrary console, socket hijacking, weakening strength of involved encryption, misrepresenting file name and type, and spoofing.

5.3 Correlating categories of vulnerabilities and impacts

Correlation or mapping between vulnerabilities and exploitation impacts is the key in formulating our attack-vulnerability-exploit model. Table 4 lists this correlation/mapping for the vulnerabilities and the impacts shown in Figure 7 and Figure 8, respectively.

Table 4: Categories of impacts caused by different categories of vulnerabilities

Vulnerability category	Impact category									
	Denial of service	Exec arb. binary code	Exec arb. shell cmd	Exec arb. web script/ HTML	Exec arb. SQL	Overw-rite arb. file	Privilege violation	Unkn-own	Misc	Total
Buffer Overflow	109	299	25	0	0	1	18	11	6	469
Integer Overflow	25	40	0	0	0	0	1	0	5	71
Format String	19	28	3	0	0	0	0	2	1	53
Race condition	7	3	0	1	0	0	6	0	1	18
SQL injection	1	0	0	2	609	0	3	3	3	621
Cross-site scripting	2	1	5	949	2	0	15	4	23	1001
Insufficient Param. input validation	14	18	8	330	2	3	86	0	36	497
Design/config error	332	94	22	160	3	52	691	11	224	1589
Unknown	111	59	15	9	0	9	101	118	48	470
Total	620	542	78	1451	616	65	921	149	347	4789

Importantly, the table is noticeably sparse. This indicates that there are patterns for vulnerabilities to cause different types of impacts. We use the dominant patterns to construct the attack model described in Section 6. Specifically, it is observed that *common/similar consequences of attacks (impact) result from exploiting the same category vulnerabilities.*

The attack model is built based on the correlation information obtained by analyzing the current vulnerability database. As more vulnerabilities and attack methods are added into the database, the model evolves and gets expanded for a larger coverage of capturing attacks.

The main patterns observed from Table 4 include:

- The first three rows of the table, *buffer overflow, integer overflow,* and *format string,* are dominated by impacts resulting in *arbitrary binary code execution* or in a *denial of service.* This matches common intuition. Nearly 63.8% (299 out of 469) of *buffer overflow* vulnerabilities cause *arbitrary binary code execution,* and 23.2% (109 out of 469) result in *denial of service.* The other impact categories are small and it can be argued that each requires an arbitrary code to be executed to be successful (e.g. a privilege violation may need also code execution ability). The number of *arbitrary binary code execution* is much larger than that of *denial of service,*

because a completely successful exploitation of buffer overflow always leads to a binary code execution; however, a partially successful exploit that crashes a single application thread may not lead to a full application crash, and the application/system may be able to recover from the thread crash. So a failed buffer overflow attack does not necessarily lead to a denial of service. Clearly, there is a close coupling between *buffer overflow* vulnerabilities and the impact of *executing arbitrary binary code.*

- *Cross-site scripting* and *SQL injection* vulnerabilities mostly contribute to *executing arbitrary web script/HTML* and *arbitrary SQL commands*, respectively, which is not surprising.
- A large number of *arbitrary script/HTML executions* are due to *cross-site scripting* and *insufficient parameter input validation* vulnerabilities (65.4% and 22.7%, respectively). The attack patterns exploiting these two vulnerability categories are different. In the *cross-site scripting* cases a web browser is attacked, while in the *insufficient parameter input validation* cases a web-based server is attacked. The observation is useful in reducing the number of states in the model.
- Interestingly, *design/config error* vulnerabilities are likely to incur *privilege violation* (691 out of 1589, or 43.5%), *denial of service* (20.9%), and *miscellaneous* (14.1%) impacts. Also, privilege violations (691 out of 921, or 75.0%) are mostly caused by vulnerabilities in the *design/config error* category, i.e. there is a close coupling between the vulnerability and the impact.

This analysis shows the existence of a close coupling between vulnerability and impact/attack consequence categories, and justifies the use of this correlation for attack model construction.

6. Attack Model

In this section we combine knowledge based on: (i) attack patterns (identified from the analysis of real attacks, see Section 4), and (ii) vulnerability and attack categories (identified from analysis of data in CVE database) to derive an attack model (a state machine). Specifically, the model construction is based on the two observations we have made: (i) common/similar consequences of attacks exploiting same category vulnerabilities; and (ii) common/similar attack-flow patterns for attacks causing same type of consequences to the victim system.

From the attack examples analyzed in Section 4, two generic types of attacks: scan based and non-scan based, are identified. In a scan-based attack, the attacker proactively locates the victim

machine and attacks it, while in a non-scan based attack, the attacker links/posts vulnerability-exploiting web pages to public websites (like forums, wikis, boards), distributes a Trojan software through the network, or installs sniffer software on a local machine, and then passively waits for a victim (or through spam emails/messages). Now we describe how the attack models are created for the two kinds of attacks.

6.1 Scan-based attacks

Based on the analysis of the real data we constructed model of scan-based attacks, which consists of four phases: (1) scan, (2) vulnerability exploitation, (3) environment setup, and (4) machine use.

Figure 9 shows the attack model for scan-based attacks. The attack model is a state machine. A circle in the figure denotes a state of the node. An arrow denotes an event that transitions the node's state. Two properties are associated with an event: the input from outside and the node's action upon the input. The properties of events are attached to the corresponding event arrows in the figure, and the two parts are separated by semicolon. In order to avoid complicating the figure, only attack-associated events are depicted.

Scan phase. Before an attack, the node performs tasks normally. The node transitions to the *being scanned* state when it receives a request (for port connecting, service, or login) and gives a negative response. (If there is no more input from the same source after a sufficiently long period of time, the node goes back to *normal* state.) A positive response of the node to an input request transitions the node from *being scanned* to *candidate* state, which means that the node has been labeled as a victim candidate by the attacker. Note that once a node enters the "candidate" state, it can not return to "normal" any more.

Vulnerability exploit phase. The attacker applies the exploit request to the node. The attack fails if the target application has been patched to remove the vulnerability (shown as dashed arrows "exploitation foiled" in Figure 9). If the exploitation succeeds, the node gives a positive response to the attacker, and the node is considered compromised. The patterns of correlation between vulnerability categories and impact categories we observe in Section 5.3 are employed for state transitions in this phase. For example, a buffer-overflow results in either denial of service or arbitrary binary code execution, and hence, there are transitions from *candidate* state to *denial of service* and *arbitrary code execution,* respectively, in Figure 9.

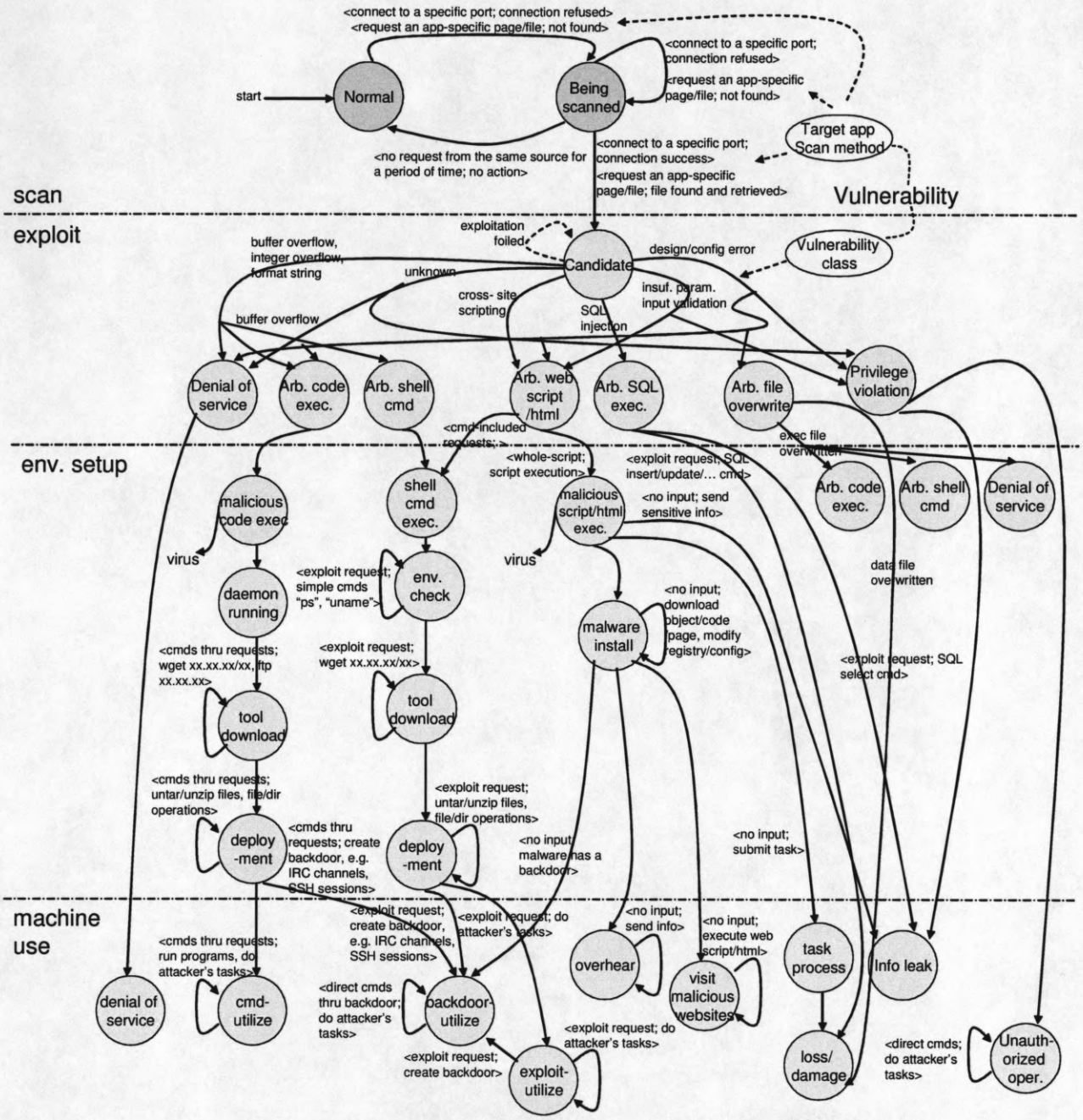


Figure 9: Attack model for a scan-based attack

Experiment setup phase. As the first successful exploitation provides limited power in manipulating the victim node (in most attacks the first successful exploitation is a test and performs a task without harming the machine), the attacker needs more powerful tools to facilitate use of the machine. The attacker directs the node to download relevant tools and code by ftp/wget. The

methods through which the tools are downloaded depend on which vulnerability exploit has taken place for the attack. Figure 9 shows that some download commands are through exploit requests, while others are through a valid request (the request is intercepted by the running daemon).

Machine use phase. After the desired tools are successfully deployed in the environment, the attacker starts programs to execute code and perform specific tasks in following steps ("cmd-utilize" and "exploit-utilize"), or sets up a backdoor for more direct manipulation of the node ("backdoor-utilize"). IRC channels, SSH sessions, and VNC sessions are candidates for backdoor connections. With backdoor connections the attacker freely controls the machine as normal use.

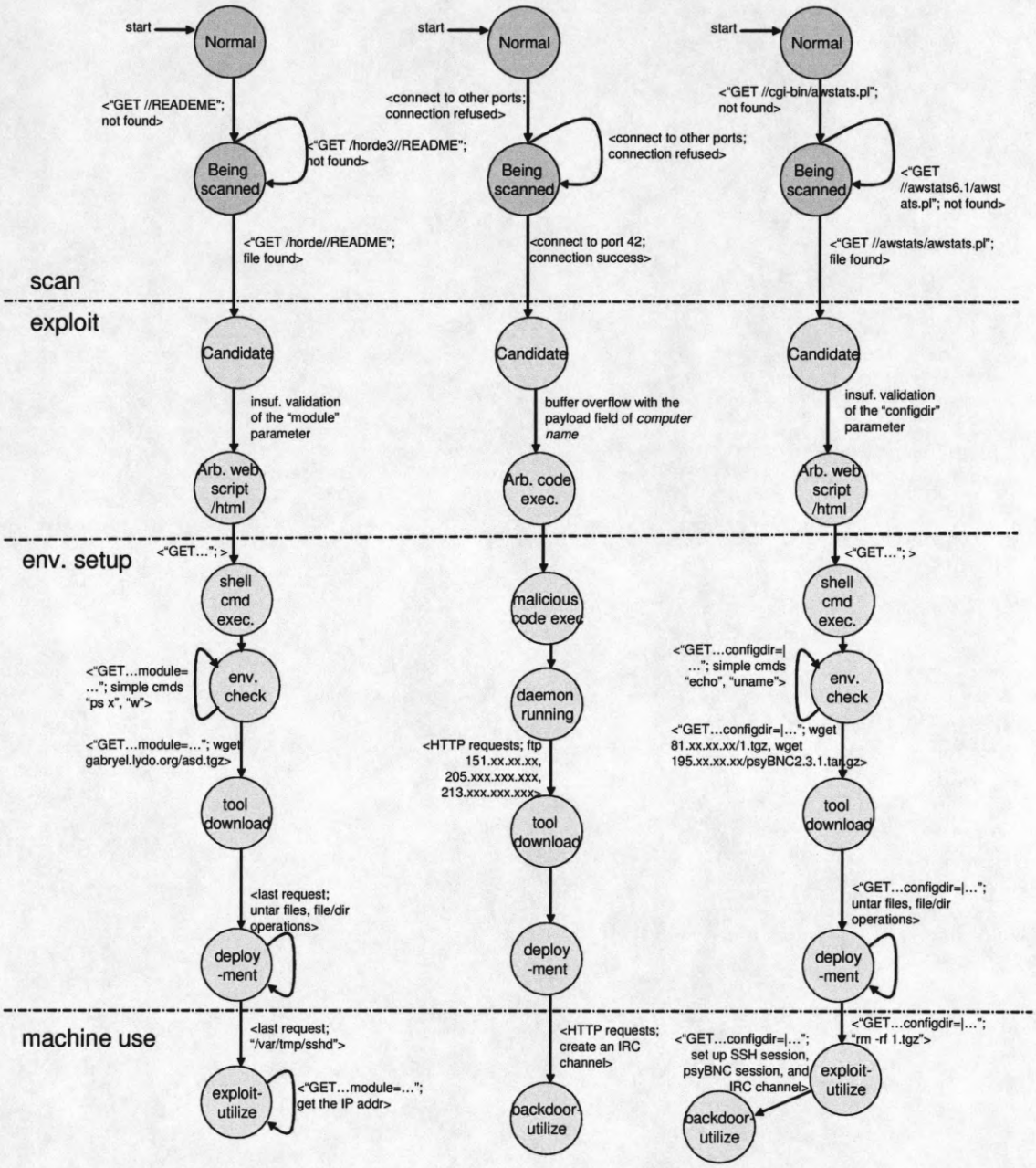
Table 5: Examples of scan-based attacks

(a) A PHP Horde attack	
Vulnerability	*Insufficient parameter input validation.* Eval() injection vulnerability in Horde Application Framework versions 3.0 before 3.0.10 and 3.1 before 3.1.1: the "module" parameter is not checked for syntax when the Horde help viewer is invoked. (CVE-2006-1491)
Scanning method	Checks if the README file of Horde exists at known paths (e.g. Horde/README) by requesting the file through HTTP protocol, e.g. *GET /horde//README*
Exploit	Supplies the "module" parameter with crafted input when requesting the help viewer, e.g. *GET //horde//services/help/?show=about&module=;".passthru("".chr(47)."sbin".chr(47)."ifconfig |grep inet");'.*
Post-exploitation actions	Downloads and executes exploit code (the "module" parameter version of the commands is not listed for simplicity): *cd /tmp ;wget gabryel.lydo.org/asd.tgz ;tar zxfv asd.tgz ;cd .asd ;cp linux /var/tmp/sshd ;/var/tmp/sshd ;/var/tmp/sshd*
(b) A WINS service attack	
Vulnerability	*Buffer overflow.* The WINS (Windows Internet Naming Service) in WinNT Server 4.0 SP 6a, NT Terminal Server 4.0 SP 6, Win2K Server SP3/SP4, and WinServer 2003 does not properly validate the computer name in a WINS packet, which allows remote attackers to execute arbitrary code or cause a denial of service (server crash) through a buffer overflow. (CVE-2004-0567)
Scanning method	Checks if the WINS service exists on a computer by connecting to TCP/UDP port 42.
Exploit	The payload field *computer name* in the WINS packet is crafted with binary instructions
Post-exploitation actions	Downloads win.png through ftp; connects to an external machine at port 50000; sets up IRC (Internet Relay Chat) channel for a backdoor entry; downloads nfs-cd.zip through ftp; executes aim.exe; downloads lusetup.exe and wincip81SR1eval.exe (for unzipping other files) through ftp
(c) An Awstats attack	
Vulnerability	*Insufficient parameter input validation.* AWStats 6.1, and other versions before 6.3, allows remote attackers to execute arbitrary script code via shell meta characters (e.g. the pipe character 'l') in the configdir parameter. (CVE-2005-0116)
Scanning method	Checks if the Awstats tool exists on a computer by sending a request, e.g. *GET //awstats/awstats.pl*
Exploit	Supplies the "configdir" parameter with crafted input when invoking the awstats script, e.g. *GET //awstats/awstats.pl?configdir=|echo ;%00*
Post-exploitation actions	Downloads an attacker script; sets up ssh sessions; downloads more files; sets up IRC (Internet Relay Chat) channel; removes the attack script (e.g. *GET //awstats/awstats.pl?configdir=|echo ;echo b_exp;cd /tmp;rm –rf 1.tgz miro;echo e_exp;%00* where the 1.tgz file is the attack script)

6.1.1 Describing attack examples using the attack model

In this section we give real examples of scan-based attacks from the NCSA data and to show how these attacks fit into the attack model introduced in Figure 9. Table 5 provides the detail data on

three attacks and the corresponding models (extracted as paths from the overall comprehensive model in Figure 9) are illustrated in Figure 10.



(a) A PHP Horde Attack (b) A WINS Service Attack (c) An Awstats Attack

Figure 10: models for the attack examples in Table 5

We use the Awstats attack (Table 5(c) and Figure 10 (c)) as the example to describe how the attack fits in the overall attack model. The victim machine transitions from *normal* to *being scanned* when the attacker tries to locate the awstats.pl file at *"//cgi-bin"* and the victim machine responds as *not*

found. After trying several paths for the file, the scan succeeds as the request of "//awstats/awstats.pl" is responded as *ok* (it matches "scanning method" in Table 5(c)), and the victim machine is now a candidate for further attack. The attacker sends simple commands (*"echo; uname"*) through the *configdir* parameter in a request (see "exploit" in Table 5(c)), in order to test vulnerability exploitation and check environment. The request exploits a vulnerability due to *insufficient parameter input validation*, and the machine transitions to *arbitrary web script/html* state, *shell command execution* state, and *environment check* state in this specified order. The post-exploit actions listed in Table 5(c) are conducted in two phases: *environment setup* and *machine use* (as shown in Figure 10 (c)). In the *environment setup* phase, the downloading of two files (1.tgz and psyBNC2.3.1.tar.gz) through an HTTP request transitions the machine into *tool download* state; following file operations install the attack script (1.tgz) and psyBNC (a backdoor program) when the machine is in *deployment* state. In *machine use* phase, the attacker issues a command ("rm –rf 1.tgz") when the machine enters *exploit-utilize* state. After the attacker sets up several SSH sessions, psyBNC sessions and an IRC channel, the machine transitions to *backdoor-utilize* state.

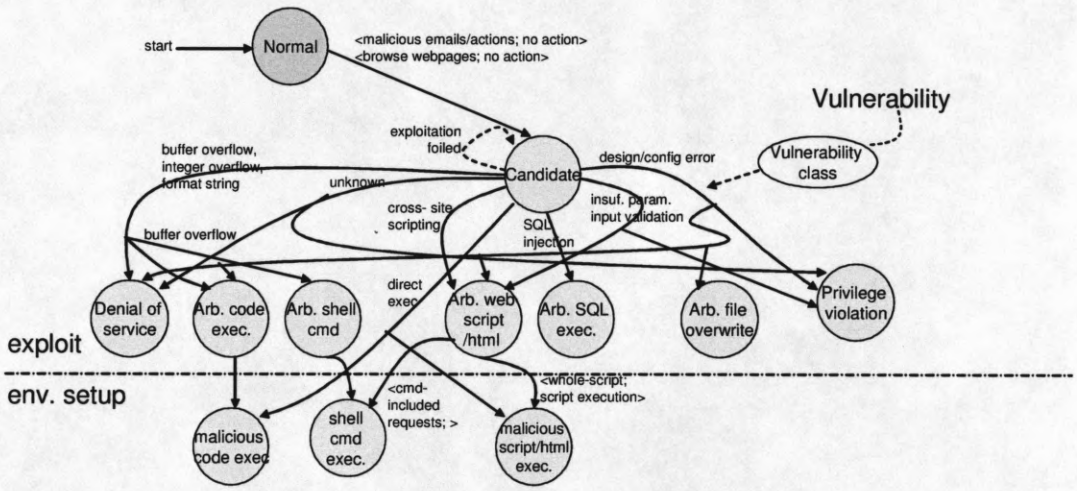


Figure 11: Attack model for a no-scan-based attack

6.2 Non-scan based attacks

For non-scan based attacks, there is no a scan stage for the machine to become a victim candidate. When a machine receives a spam email or browses a page that contains a link to a malicious webpage, the machine becomes a victim candidate. The other phases for non-scan based attacks, including attack preparation, exploit, environment setup, and machine use, are similar to scan-based attacks.

Apart from the different impacts resulting from different classes of vulnerabilities, a candidate can directly change into the state of *malicious code execution* or *malicious script/html execution*, because a Trojan software or a malicious webpage may be launched directly by the user without exploiting any vulnerability. However, all kinds of vulnerabilities can also be exploited if the user visited a page with a script sending requests for such exploits.

Table 6: Examples of non-scan attacks

(a) A Remote xserver logging attack	
Vulnerability	*Design/config error.* Keystroke input of a user to the same remote Xserver can be logged by another user.
Scanning method	No need for scanning. The attacker entered the system as a valid user
Exploit	The attacker downloaded keystroke sniffer and installed it on the machine. When the root logged in the root password was sent to the attacker by email.
Post-exploitation actions	Logs in the machine as root.
(b) A phishing attack	
Vulnerability	*Design/config error.* Browser cookie for a website is sent with all requests as long as the destination is the website.
Scanning method	No need for scanning. The victim user clicks the bad link.
Exploit	Valid cookie is sent with malicious requests to the target website, which considers the requests are from the authenticated user.
Post-exploitation actions	Disables notification setting; associates a checking account; transfers money from the user account into the new checking account; deletes the checking account; restores notification setting.

6.2.1 Describing attack examples using the attack model

Real examples of non-scan based attacks are given here to show how these attacks fit into the attack model introduced in Figure 11. Table 6 provides two attacks and the corresponding models (extracted as paths from the overall comprehensive model in Figure 11) are illustrated in Figure 12.

We use the remote xserver logging attack as the example for description in this section. The target machine B is first in the *normal* state. The attacker is a valid user of another machine A (e.g. a gateway node of an intra-network) and downloads/installs an xserver keystroke logger on machine A, which makes machine B a victim candidate (the root of machine B potentially logs in machine B through machine A). The design error of the xserver on machine A allows a user to log keystroke from another root user logging into into machine B via an X channel (GUI on Unix/Linux) through machine A. Then the installed software sends the root password to the attacker via email, and machine B transitions to *privilege violation* state. The attacker now logs in machine B as root and machine B is in *unauthorized operation* state.

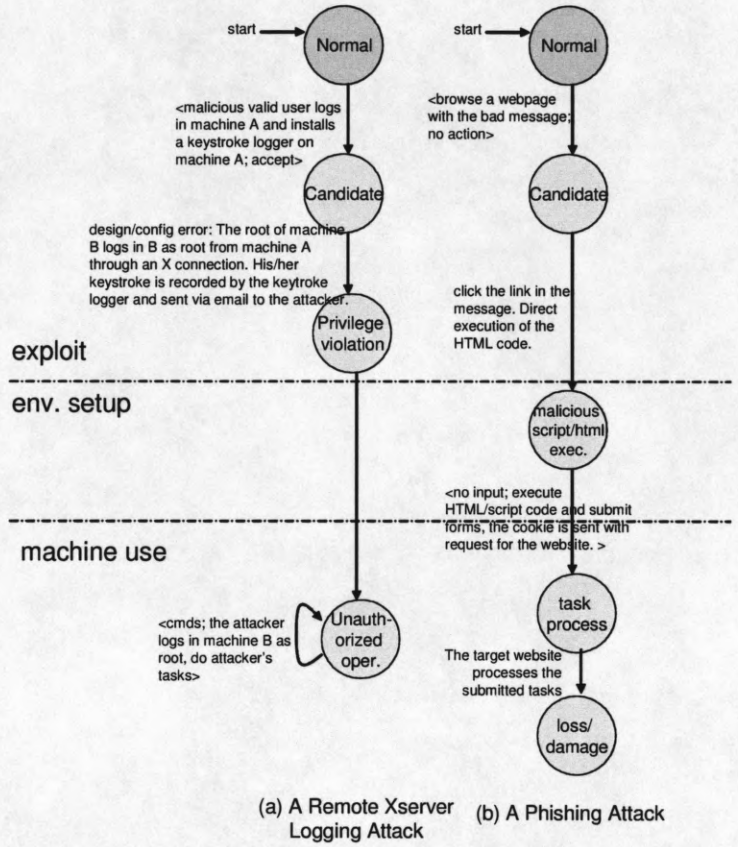


Figure 12: models for the attack examples in Table 6

7. Conclusion

This paper proposes a methodology to analyze attacks based on the observations that (i) a class of vulnerabilities usually lead to one or several categories of exploitation impacts, and (ii) a category of vulnerability exploitation scenarios are usually associated with characteristic attack flow patterns. The correlation/mapping between vulnerabilities and exploitation impacts can be obtained by analyzing vulnerability database; the attack flow patterns can be achieved by studying real attack cases. Then the two can be combined to create comprehensive attack models. Future research could address the use of such models to build defenses against new attacks.

References

[1] Aslam, T., Krsul, I., Spafford, E. Use of A Taxonomy of Security Faults. *In 19th National Information Systems Security Conference Proceedings.* Baltimore, Maryland, 1996.

[2] S. Whalen, M. Bishop, and S. Engle, Protocol Vulnerability Analysis, *Technical Report CSE-2005-4,* Dept. of Computer Science, University of California at Davis, Davis, CA 95616-8562 (May 2005).

[3] C. Vanden Berghe, J. Riordan, and F. Piessens, A Vulnerability Taxonomy Methodology applied to the Web Services, Proceedings of the 10th Nordic Workshop on Secure IT Systems (NordSec 2005) (Lipmaa, H. and Gollmann, D., eds.), pp. 49-62, 2005.

[4] Krsul, Ivan, "Software Vulnerability Analysis," PhD Thesis, Coast TR 98-09, Department of Computer Science, Purdue University, 1998.

[5] S. Chen, Z. Kalbarczyk, J. Xu and R. K. Iyer, A Data-Driven Finite State Machine Model for Analyzing Security Vulnerabilities, Proc. IEEE International Conference on Dependable Systems and Networks, DSN'03, San Francisco, CA, June 22-25, 2003.

[6] Cohen, Fred, "Information System Attacks: A Preliminary Classification Scheme," *Computers & Security*, Volume 16, Number 1, Pages 29-46, 1997.

[7] Cukier, M., Berthier, R., Panjwani, S., Tan, S., A statistical analysis of attack data to separate attacks, *In Proc. Dependable Systems and Networks,* Philadelphia, 2006.

[8] Perry, T. and P. Wallich, Can Computer Crame be Stopped?, *IEEE Spectrum,* vol. 21, num 5, May 1984.

[9] CERT/CC statistics, http://www.cert.org/stats/cert_stats.html

[10] Robert A. Martin, Managing Vulnerabilities in Networked Systems, *IEEE Computer*, vol 34 issue 11, Nov 2001.

[11] http://cve.mitre.org/

[12] http://www.securityfocus.com/archive/1

[13] U. Lindqvist and E. Jonsson. How to Systematically classify computer security intrusions. *Proc. Of IEEE Symposium on Security and Privacy,* Oakland, CA, May 1997.

[14] Katrina Teipenyuk, Brian Chess, & Gary McGraw. Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors. *IEEE Security & Privacy*, November/December 2005.

[15] Cisco Systems. White paper: NetFlow services and applications. Available at http://www.cisco.com.

[16] Minho Sung, Markus Haas and Jun Xu, "Analysis of DoS attack traffic data", *Forum of Incident Response and Security Teams (FIRST)*, Hawaii, June 2002.

[17] James J. Treinen and Ramakrishna Thurimella. A Framework for the Application of Association Rule Mining In Large Intrusion Detection Infrastructures, *Recent Advances in Intrusion Detection (RAID),* Hamburg, Germany, Sep 2006.

[18] V. Paxson, Bro: A System for Detecting Network Intruders in Real-Time, *Computer Networks*, 31(23-24), pp. 2435-2463, 14 Dec. 1999.

[19] Denning, D., et al. "A Prototype IDES: A Real Time Intrusion Detection Expert System." Computer Science Laboratory, SRI International, August 1987.

[20] Roger A. Grimes, Honeypots for Windows., Chap 11, *Apress L. Press.*, February 2005.

[21] McGrew, R., Vaughn, R., Experiences with Honeypot Systems: Development, Deployment, and Analysis. *Hawaii International Conference on System Sciences*, Jan. 2006.

[22] Jonathon T. Giffin, Model-based intrusion detection system design and evaluation. Ph.D. thesis, *University of Wisconsin-Madison*, Madison, Wisconsin, August 2006.

[23] Ruiu, Dragos. Cautionary Tales: Stealth Coordinated Attack HOWTO. *http://www.nswc.navy.mil/ISSEC/CID/Stealth_Coordinated_Attack.html*, 1999.

[24] Tidwell, T., Larson, R., Fitch, K., and Hale, J. Modeling Internet Attacks, *Proc. IEEE Workshop on Information Assurance and Security,* United States Military Academy, West Point, NY, June 2001.

[25] Sandeep Kumar and Eugene Spaord. An Application of Pattern Matching in Intrusion Detec-tion. Technical Report 94-013, Purdue University, Department of Computer Sciences, March1994.

[26] Templeton, Steven J. and Karl Levitt, "A Requires/Provides Model for Computer Attacks", Proc of the New Security Paradigms Workshop 2000, Cork Ireland, Sep 2000.

[27] C.R. Ramakrishnan and R. Sekar, Model-based vulnerability analysis of computer systems, in: 2nd Internat. Workshop on Verification, Model Checking and Abstract Interpretation, 1998.

[28] Srikanth Kandula, Sankalp Singh and Dheeraj Sanghi, Argus - A Distributed Network Intrusion Detection System, USENIX SANE, Maastricht, Netherlands, May 2002.

[29] Tripwire Intrusion Detection System 1.3 for LINUX User Manual, http://www.cosmic-ray.org/miscfiles/idsl_1_3.pdf

[30] http://labrea.sourceforge.net/

[31] Alex Stamos, Zane Lackey, Attacking AJAX Applications: Vulns 2.0 in Web 2.0, *Technical Presentation, ISecPartners LLC,* http://www.isecpartners.com/files/Attacking_AJAX_Applications-Toorcon2006.pdf, Sep 2006.