**CPS 633 Project**

**Introduction to Security**

Warren Pinto [wpinto@ryerson.ca](mailto:wpinto@ryerson.ca),

                                Thusitharan Paramsothy [thusitharan.paramsot@ryerson.ca](mailto:thusitharan.paramsot@ryerson.ca),

              Sundar Murugesan [mmuruges@ryerson.ca](mailto:mmuruges@ryerson.ca),

                                      Shawn Quach [shawn.quach@ryerson.ca](mailto:shawn.quach@ryerson.ca)  
  
**Abstract :-** *The main purpose of this report is to design an algorithm that identifies, replicates, and prevents an attack to a system using the Case-Based Reasoning (CBR) approach.  This will help us expand our base cases in identifying threats through the means of other proposed solutions of an existing problem. Using our base-case, we will then try to implement an algorithm to detect and prevent future attacks that is similar to the initial base-case. In the initial base case, we will be looking at how an attacker can effectively gain root access control over a File Transfer Protocol (FTP) server using an exploit in the “Wu-FTP daemon”. To identify the attack in the initial base case, various logging techniques will be used. Some of the programs used for the proposed initial base case includes Snort and TCPFlow. The tools that we will be implementing are Snort.*

**Introduction**

**Section 1**

The use of personal computers in today’s world has increased significantly over the years of its conception.  Consumers such as business corporations, students, to household owners use computers for a wide margin of work.  But like with everything considered valuable to a person, one must know the limitations of its security and the possibilities of it being attacked or breached.  In order to obtain knowledge within this area, users must expand their knowledge on the topic of computer security.  To do this, an assignment that provides the ability for one to develop his/her forensic and analysis skills would be essential. For this case, the goal of this research project is to review the given scan from the Honeynet Project and apply our knowledge in developing algorithms in correspondence to the scenario using a Case-Based Reasoning (CBR) approach.

Case Based Reasoning is a topic from Artificial Intelligence where new problems are solved from a studied problem. CBR techniques can be seen across many industries that need applications to make informed decisions for a problem. Taking strategy based games for example, we can see a CBR technique used in a virtual game of chess where the base case would have a collection of cases of simulated games that would be useful for the computer to utilize against its opponent. CBR goes through the following four stages: *retrieve*, *reuse*, *revise* and *retain*.  A new problem is matched against cases in the case base and one or more similar cases are retrieved. A solution suggested by the matching cases is then reused and tested for success. Unless the retrieved case is a close match the solution will probably have to be revised producing a new case that can be retained. In this project CBR played a major role in solving new problems that arose in its development. The process started with identifying the pattern of the attack by viewing snort rules to see where exactly our system was compromised which eventually lead us to making an algorithm for our base case. More details about Scan 19, the procedure followed, and important background concepts will be discussed in the remainder of this report.

**Buffer Overflow**

To define what a buffer overflow is, we must first break down the term. First off, a buffer in computer science terms is basically temporary memory used to store input and output data. When combined with the word overflow, it now defines the situation where the amount of data attempting to be stored into the temporary is more than intended. Now because there isn’t enough space to contain all the data, the remaining amount gets stored into adjacent buffers if present. When looking at this concept in terms of a computer security, it can be used as an attack where the extra data may contain lines of code instructed to harm the targeted system in whatever way possible.

**Rootkits**

Rootkits are a set or collection of tools that an attacker places after gaining user-level access in the victim’s computer without the victim being aware of it. These set of tools enables the attacker to gain administrator-level access and thus demonstrates a breach of security. A rootkit can consist of spywares and programs such as monitoring actions, trapdoors, altering log files, attacks to other computers, etc. Attackers use rootkits because of its subtle operations and because it is capable of attacking other computers in the system; the damage’s range can spread vastly.

**Syslog**

Syslog is a protocol in UNIX and LINUX systems that allows machines to generate event notification messages and transmit these messages over the network and possibly towards a specific Syslog Server. Each message consists of several attributes which are called the PRI, HEADER and MSG. PRI is broken down into two parts: Severity Codes and Facility Codes which can be seen in the example below.

**a) Severity Codes:** Severity codes is the severity of the message being made.

Syslog Severity Code

|  |  |
| --- | --- |
| Numerical Code | Severity |
| 0 | Emergency: system is unusable |
| 1 | Alert: action must be taken immediately |
| 2 | Critical: critical conditions |
| 3 | Error: error conditions |
| 4 | Warning: warning conditions |
| 5 | Notice: normal but significant condition |
| 6 | Informational: informational messages |
| 7 | Debug: debug-level messages |

    b) **Facility Codes:** The application that generates the log messages.  
  
Syslog Facility Code

|  |  |
| --- | --- |
| Numerical Code | Facility |
| 0 | kernel messages |
| 1 | user-level messages |
| 2 | mail system |
| 3 | system daemons |
| 4 | security/authorization messages |
| 5 | messages generated internally by syslogd |
| 6 | line printer subsystem |
| 7 | network news subsystem |
| 8 | UUCP subsystem |
| 9 | clock daemon |
| 10 | security/authorization messages |
| 11 | FTP daemon |
| 12 | NTP subsystem |
| 13 | log audit |
| 14 | log alert |
| 15 | clock daemon |
| 16 | local use 0 |
| 17 | local use 1 |
| 18 | local use 2 |
| 19 | local use 3 |
| 20 | local use 4 |
| 21 | local use 5 |
| 22 | local use 6 |
| 23 | local use 7 |

**Tools used**

**Snort IDS**

Snort is an open source Intrusion Detection System (IDS). It is able to perform real-time traffic analysis and packet loggings in an operational network. By following a set of rules, the IDS can also be used to track a number of attacks or vulnerabilities such as buffer-overflows, backdoors, exploits, Denial of Service (DDoS) attacks, and many more. The typical uses of the program consist of packet sniffing, packet logging, and network intrusion prevention systems.

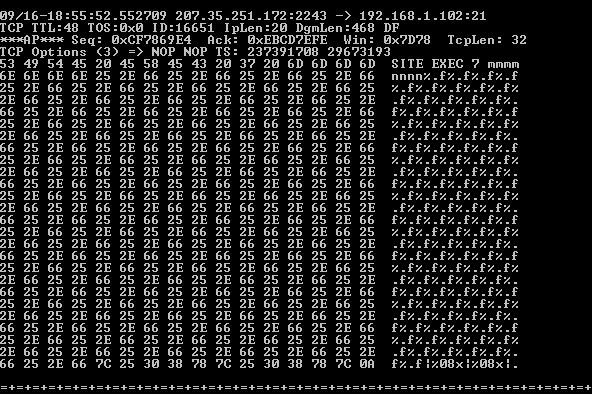
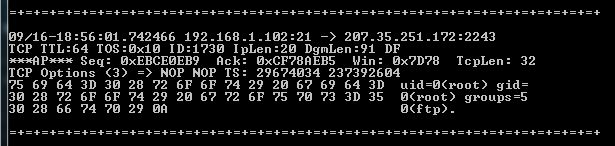
Commands available with snort:

Refer to appendix

**Attack Patterns:-**

Looking from the Snort log files, we can determine the following attack patterns which were used to gain access and attack the FTP server. Initially, the server requested a login name and password in which the attacker supplied with and successfully logged in as a guest user. The attacker then exploited the SITE EXEC vulnerability by executing the SITE EXEC command. Following the command, the attacker supplied the arguments with several %.f characters.

At time 18.55.52.235847, Snort detected a SITE EXEC command, followed by several %.f characters

The server successfully completed the command, and as a result, the attacker was given root privileges. The snort rules captured the following data from the newdat3.log file:

*09/16-18:55:59.485710  [\*\*] [1:344:0] FTP-SITE-EXEC-EXPLOIT [\*\*] [Classification: Attempted Administrator Privilege Gain] [Priority: 1] {TCP} 207.35.251.172:2243 -> 192.168.1.102:21*

*09/16-18:56:01.742466  [\*\*] [1:498:8] INDICATOR-COMPROMISE id check returned root [\*\*] [Classification: Potentially Bad Traffic] [Priority: 2] {TCP} 192.168.1.102:21 -> 207.35.251.172:2243*

Following the root access privilege, the attacker installed rootkits Zer0.tar.gz, copy.tar.gz and ooty.tar.gz into the server.

Looking at the log entries, we can also determine the time of the attack and the IP address of the attacker.

**commands used:**  
  
snort -r C:\Snort\log\newdat3.log -c C:\Snort\etc\snort.conf -l C:\Snort\log -A console > C:\Snort\log\newdatXX.txt  
  
snort -dvqr C:\Snort\log\newdat3.log > C:\Snort\log\newdat3.txt  
  
snort -vdr C:\Snort\log\slog2.log > C:\Snort\log\commands\_used.txt

**CBR EXPRESS CASE**   
BEGIN  CASE 1  
TITLE  
    Ftp-Site-Exec exploit   
  
DESCRIPTION  
    User gains root access privileges through SITE EXEC command  
  
QUESTIONS  
    Is user a GUEST?  
    ANSWER: **YES**  
    SCORING; **(default)**  
Is EXEC COMMAND entered into command line  
    ANSWER: **YES**  
    SCORING:  **(default)**  
Several %f arguments entered, to exploit?  
    ANSWER: **YES**  
    SCORING: **(default)**  
Root access granted?  
    ANSWER: **YES**  
    SCORING: **-**  
ACTIONS  
    Print out similarities from the initial base case.  
END CASE

File: Initial\_case.bat

ECHO FTP SITE EXEC BUFFER findstr "SITE" C:\Snort\log\newdat3.log

set RESULT1=%ERRORLEVEL%

if %RESULT1% == 0 (Echo true)

ECHO FTP SITE EXEC LOG

findstr "SITE" C:\Snort\log\newdatXX.txt

set RESULT2=%ERRORLEVEL%

if %RESULT2% == 0 (Echo true)

ECHO ROOT KIT COMMANDS

findstr ".tar" C:\Snort\log\commands\_used.txt

set RESULT3=%ERRORLEVEL%

if %RESULT3% == 0 (Echo true)

PAUSE

The above code searches through the log file for the initial base case. If the initial base case succeeds, the program will return true. If there are unique log entries in the log file, then output to the initial base case file and thus expanding the initial base case. For example, a user develops his/her own syslog entries. The user’s syslog entries are then compared with the initial base case log entries using the initial\_case.bat file, and if there is a match, output a SITE EXEC exploit. If there are unique log entries in addition to the base case then add it to the initial base case log file.

**Conclusions and Future Work**

In this project we took common features of our given scan and designed a CBR algorithm. We then organized our algorithm into attributes that we thought were important. These attributes included buffer overflow, site exec command and rootkits.  
  
One of the major problems we had during the beginning of this project was downloading Snort into our Ubuntu system using VMware player. A lot of time and research was spent on finding credible installation instructions. However, no solution was found. Some common errors   
included connection errors during download which resulted in missing some crucial libraries. If more time was given during the project we would have emailed technical support to figure out exactly how to find a solution to downloading it in our system.  
  
Another issue that we could not finish due to time constraints was improving our base case. If more time was put on researching scans we could have found a big list of scans to improve our product.

**Appendix**

USAGE: snort [-options] <filter options>  
     snort /SERVICE /INSTALL [-options] <filter options>  
     snort /SERVICE /UNINSTALL  
     snort /SERVICE /SHOW  
Options:  
  
      -A         Set alert mode: fast, full, console, test or none  (alert file alerts only)  
      -b         Log packets in tcpdump format (much faster!)  
      -B <mask>  Obfuscated IP addresses in alerts and packet dumps using CIDR mask  
      -c <rules> Use Rules File <rules>  
      -C         Print out payloads with character data only (no hex)  
      -d         Dump the Application Layer  
      -e         Display the second layer header info  
      -E         Log alert messages to NT Eventlog. (Win32 only)  
      -f         Turn off fflush() calls after binary log writes  
      -F <bpf>   Read BPF filters from file <bpf>  
      -G <0xid>  Log Identifier (to uniquely id events for multiple snorts)  
      -h <hn>    Set home network = <hn>  
                 (for use with -l or -B, does NOT change $HOME\_NET in IDS mode)  
      -H         Make hash tables deterministic.  
      -i <if>    Listen on interface <if>  
      -I         Add Interface name to alert output  
      -k <mode>  Checksum mode (all,noip,notcp,noudp,noicmp,none)  
      -K <mode>  Logging mode (pcap[default],ascii,none)  
      -l <ld>    Log to directory <ld>  
      -L <file>  Log to this tcpdump file  
      -n <cnt>   Exit after receiving <cnt> packets  
      -N         Turn off logging (alerts still work)  
      -O         Obfuscate the logged IP addresses  
      -p         Disable promiscuous mode sniffing  
      -P <snap>  Set explicit snaplen of packet (default: 1514)  
      -q         Quiet. Don't show banner and status report  
      -r <tf>    Read and process tcpdump file <tf>  
      -R <id>    Include 'id' in snort\_intf<id>.pid file name  
      -s         Log alert messages to syslog  
      -S <n=v>   Set rules file variable n equal to value v  
      -T         Test and report on the current Snort configuration  
      -U         Use UTC for timestamps  
      -v         Be verbose  
      -V         Show version number  
      -W         Lists available interfaces. (Win32 only)  
      -X         Dump the raw packet data starting at the link layer  
      -x         Exit if Snort configuration problems occur  
      -y         Include year in timestamp in the alert and log files  
      -Z <file>  Set the performonitor preprocessor file path and name  
      -?         Show this information  
<Filter Options> are standard BPF options, as seen in TCPDump  
Longname options and their corresponding single char version  
 --logid <0xid>                  Same as -G  
 --perfmon-file <file>           Same as -Z  
 --pid-path <dir>                Specify the directory for the Snort PID file  
 --snaplen <snap>                Same as -P  
 --help                          Same as -?  
 --version                       Same as -V  
 --alert-before-pass             Process alert, drop, sdrop, or reject before pass, default is pass before alert, drop,...  
 --treat-drop-as-alert           Converts drop, sdrop, and reject rules into alert rules during startup  
 --treat-drop-as-ignore          Use drop, sdrop, and reject rules to ignore session traffic when not inline.  
 --process-all-events            Process all queued events (drop, alert,...), default stops after 1st action group  
 --enable-inline-test            Enable Inline-Test Mode Operation  
 --dynamic-engine-lib <file>     Load a dynamic detection engine  
 --dynamic-engine-lib-dir <path> Load all dynamic engines from directory  
 --dynamic-detection-lib <file>  Load a dynamic rules library  
 --dynamic-detection-lib-dir <path> Load all dynamic rules libraries from directory  
 --dump-dynamic-rules <path>     Creates stub rule files of all loaded rules libraries  
 --dynamic-preprocessor-lib <file>  Load a dynamic preprocessor library  
 --dynamic-preprocessor-lib-dir <path> Load all dynamic preprocessor libraries from directory  
 --dynamic-output-lib <file>  Load a dynamic output library  
 --dynamic-output-lib-dir <path> Load all dynamic output libraries from directory  
 --pcap-single <tf>              Same as -r.  
 --pcap-file <file>              file that contains a list of pcaps to read - read mode is implied.  
 --pcap-list "<list>"            a space separated list of pcaps to read - read mode is implied.  
 --pcap-loop <count>             this option will read the pcaps specified on command line continuously.  
                                 for <count> times.  A value of 0 will read until Snort is terminated.  
 --pcap-reset                    if reading multiple pcaps, reset snort to post-configuration state before reading next pcap.  
 --pcap-show                     print a line saying what pcap is currently being read.  
 --exit-check <count>            Signal termination after <count> callbacks from DAQ\_Acquire(), showing the time it  
                                 takes from signaling until DAQ\_Stop() is called.  
 --conf-error-out                Same as -x  
 --enable-mpls-multicast         Allow multicast MPLS  
 --enable-mpls-overlapping-ip    Handle overlapping IPs within MPLS clouds  
 --max-mpls-labelchain-len       Specify the max MPLS label chain  
 --mpls-payload-type             Specify the protocol (ipv4, ipv6, ethernet) that is encapsulated by MPLS  
 --require-rule-sid              Require that all snort rules have SID specified.  
 --daq <type>                    Select packet acquisition module (default is pcap).  
 --daq-mode <mode>               Select the DAQ operating mode.  
 --daq-var <name=value>          Specify extra DAQ configuration variable.  
 --daq-dir <dir>                 Tell snort where to find desired DAQ.  
 --daq-list [<dir>]              List packet acquisition modu

**References**

**Beale, J., & Foster, J. C. (2003). *Snort 2.0 intrusion detection*. Rockland: Syngress ;.**

**CBR - Case Based Reasoning. (n.d.). *CBR - Case Based Reasoning*. Retrieved November 29, 2012, from** [**https://engineering.purdue.edu/~engelb/abe565/cbr.html**](https://engineering.purdue.edu/%7Eengelb/abe565/cbr.html)

**Clark, M. (n.d.). Scan 19. *The Honeynet Project*. Retrieved November 30, 2012, from** [**http://old.honeynet.org/scans/scan19/scan19.html**](http://old.honeynet.org/scans/scan19/scan19.html)

**Hansen, J. B. (2003). *The Hacker's Handbook*. Boca Raton: C R C Press LLC.**

**Iqbal, N., Ashraf, M. (2006). *Evaluation of jCOLIBRI.* Master Thesis Report: Malladien University Institute**

**McDowell, M. (2006, January 25). US-CERT Tip ST06-001 - Understanding Hidden Threats: Rootkits and Botnets. *US-CERT - United States Computer Emergency Readiness Team*. Retrieved November 30, 2012, from** [**http://www.us-cert.gov/cas/tips/ST06-001.html**](http://www.us-cert.gov/cas/tips/ST06-001.html)

**Rehman, W. (2003, March 25). Introduction to Syslog Protocol - MonitorWare. *MonitorWare - An Integrated Solution - MonitorWare*. Retrieved November 30, 2012, from** [**http://www.monitorware.com/common/en/articles/syslog-described.php**](http://www.monitorware.com/common/en/articles/syslog-described.php)

**Rouse, M. (n.d.). What is buffer overflow? - Definition from WhatIs.com. *Information Security information, news and tips - SearchSecurity.com*. Retrieved November 30, 2012, from** [**http://searchsecurity.techtarget.com/definition/buffer-overflow**](http://searchsecurity.techtarget.com/definition/buffer-overflow)

**Rouse, M. (n.d.). What is rootkit? - Definition from WhatIs.com. *Midmarket IT Security information, news and tips - SearchMidmarketSecurity.com*. Retrieved November 30, 2012, from** [**http://searchmidmarketsecurity.techtarget.com/definition/rootkit**](http://searchmidmarketsecurity.techtarget.com/definition/rootkit)

**Tipton, H. F., & Krause, M. (2003). Controlling FTP:  Providing Secured Data Transfers. *Information security management handbook* (4th ed., pp. 21-42). Boca Raton: Auerbach publications.**