# Detecting Distributed SQL Injection Attacks in a Eucalyptus Cloud Environment

Alan Kebert, Bikramjit Banerjee, Glover George,
Juan Solano
School of Computing
The University of Southern Mississippi
Hattiesburg, MS 39402, USA
Alan.Kebert@eagles.usm.edu

Wanda Solano
National Center for Critical Information
Processing and Storage
National Aeronautics and Space Administration
Stennis Space Center, MS 39529, USA
Wanda.m.solano@nasa.gov

*Abstract*—**Cloud computing environments offer malicious users the ability to spawn multiple instances of cloud nodes that are similar to virtual machines, except that they can have separate external IP addresses. In this paper we demonstrate how this ability can be exploited by an attacker to distribute his/her attack, in particular SQL injection attacks, in such a way that an intrusion detection system (IDS) could fail to identify this attack. To demonstrate this, we set up a small private cloud, established a vulnerable website in one instance, and placed an IDS within the cloud to monitor the network traffic. We found that an attacker could quite easily defeat the IDS by periodically altering its IP address. To detect such an attacker, we propose to use *multi-agent plan recognition,* where the multiple source IPs are considered as different agents who are mounting a collaborative attack. We show that such a formulation of this problem yields a more sophisticated approach to detecting SQL injection attacks within a cloud computing environment.**

**Tracks: Computer Security (Hacking techniques and related issues, Intrusion detection systems, Malware analysis, Intrusion detection); Security Applications (Cloud computing security)**

***Keywords—cloud computing; Distributed Attack; Eucalyptus; SNORT; Havij; OSSIM; Multi-agent plan recognition***

## I. INTRODUCTION

Cloud computing offers new opportunities for software distribution, resource allocation, convenience, and information storage and security for users, but it also creates new opportunities for malicious users to penetrate security layers and damage, destroy or steal data of other users. One advantage that a cloud computing environment offers to malicious users is the ability to spawn multiple instances of cloud nodes that are similar to virtual machines, except that they can have separate external IP addresses. In this paper we demonstrate how this ability can be exploited by an attacker to distribute his/her attack, in particular SQL injection attacks, in such a way that an intrusion detection system (IDS) could fail to identify this attack. To demonstrate this, we set up a small private cloud using the Eucalyptus [10] cloud environment, established a vulnerable website in one instance, and placed an IDS (open source OSSIM [11]) within the cloud to monitor the network traffic. We found that an attacker, using a freely available SQL injection tool (Havij) could quite easily defeat OSSIM by periodically altering its IP address, i.e., by hopping from one instance to another in the cloud.

To detect such an attacker, we propose to use *multi-agent plan recognition* [1][2][4][5], where the multiple source IPs are considered as different agents who are mounting a collaborative attack. We show that such a formulation of this problem yields a more sophisticated approach to detecting SQL injection attacks within a cloud computing environment.

## II. RELATED WORK

In the past, very little work has been done to study security issues and strategies in a cloud computing environment. A paper titled ñDigital Forensics for Eucalyptusò [9] considered security vulnerabilities in a Eucalyptus cloud, and our work can be considered as an extension or a continuation of that work, since we not only address exploitation of some vulnerabilities of Eucalyptus cloud, but also how to detect a resulting attack, where existing IDS fail.

SQL injection continues to be a threat and is discussed in depth in "A classification of SQL-injection attacks and countermeasuresò [3]. Although multiple methods exist to prevent or detect SQL injection attempts, these methods tend to focus on single actions. It can be difficult to differentiate a single action of an attack from normal traffic, so Security information and event management programs (SIEMs) try to correlate multiple activities with the plan of an attacker [6]. SIEM directives typically look for a pattern of activity from a single user to increase the reliability of an alert, but do not consider whether the actions of multiple agents have collectively achieved a malicious goal.

Multi-agent plan recognition [1][2][4][5] (MAPR) has been formalized and studied recently in abstract and theoretical settings, and to the best of our knowledge it has not been applied to any realistic cyber-security problem. Hence in this respect our work constitutes the first practical application of MAPR.

## III. Description of Setting

In this section we describe how the various components of our system are set up, and how they operate. In succession, we will describe the Eucalyptus cloud setup that we used, the Havij SQL injection tool and the network traffic sniffer Snort, which is used as a sensor by the security event manager OSSIM to generate its alerts. Finally we describe how a simple strategy of switching source IP address can defeat OSSIM.

### A. *Eucalyptus Cloud*

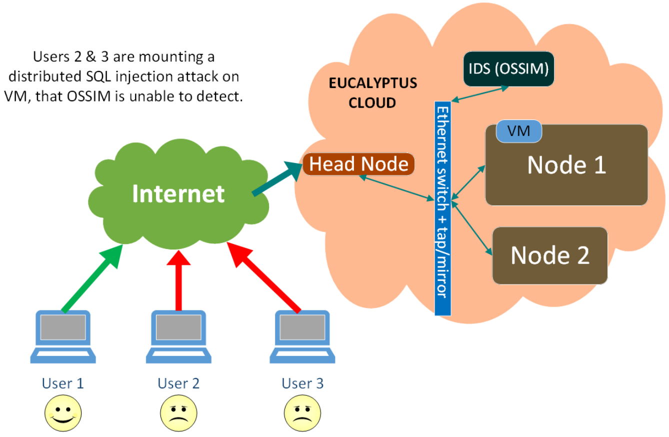
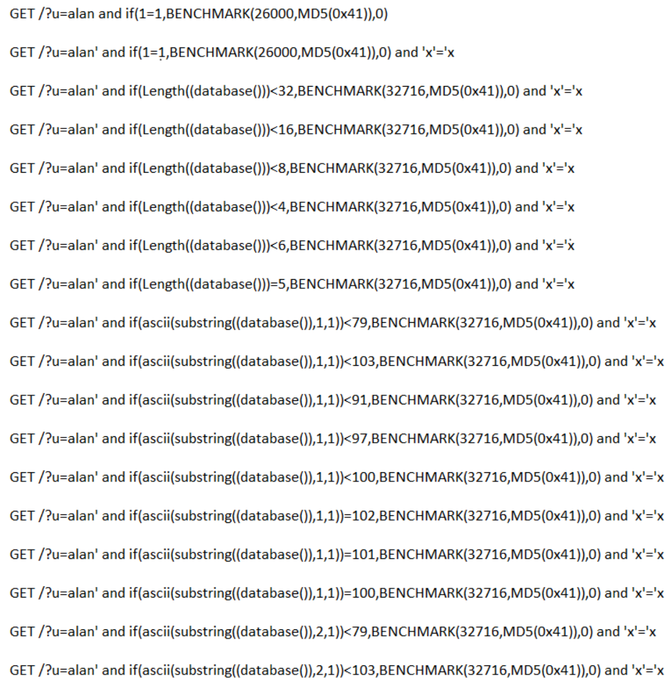

Users 2 & 3 are mounting a distributed SQL injection attack on VM, that OSSIM is unable to detect.
EUCALYPTUS CLOUD
IDS (OSSIM)
VM
Node 1
Node 2
Head Node
Ethernet switch + tap/mirror
Internet
User 1
User 2
User 3

Fig. 1. The Eucalyptus Cloud Environment

The cloud environment on which this work is based is shown in Fig. 1. It contains three nodes, the head node ï the manager of all communication with the external world ï and two other nodes that offer various computational and storage resources to users. The communication between the head node and the other nodes are via an ethernet switch. An IDS (OSSIM) sniffs all packets passing through this switch. This gives OSSIM a vantage point to monitor any external attack on resources within the secondary nodes. In particular, we establish a vulnerable website within a VM in node 1. Fig. 1 also shows users outside the cloud accessing the cloud resources through the head node. Our attack computers were located outside the Eucalyptus cloud, but still able to compromise the database inside the cloud node.

### B. *Havij*

In order to demonstrate an attack we used a program called Havij. Havij is a freely available SQL injection tool. SQL injection is the process of inserting arbitrary SQL code into a form whose input is queried against a SQL database. The form expects a user input, such as a username field on a login page, but if the text is not carefully sanitized a malicious user may place SQL commands into the field and cause the database to execute unintended commands. Havij facilitates this sort of activity by discovering the important field names needed for many SQL commands: database names, table names, and the columns of the tables. Havij can also reveal the contents of an unsecured database. It does this by first issuing a series of if-statements that test the length of field names, and then test the numerical value of the ascii characters representing individual characters of field names. Havij cannot ask the SQL database for the values directly, so it uses these comparisons to perform a binary search against a table of ascii numerical values. Each comparison will usually return zero immediately if false, but if true then an expensive MD5 benchmarking will be performed on a given string whose runtime will be reported to Havij. Based on this runtime, Havij can detect the binary outcome of the comparison. Fig. 2 shows a partial example of this process. The statements containing ñ`if (Length`ò are part of a single binary search to determine the length (in this example 5) of the name of the database. The subsequent statements containing ñ`if (ascii(substring`ò attempt to find the 5 characters one by one. The last statement of the form ñ`if (ascii(substring((database()),x,1))=y,BENCHMARK`é ò that contains `=y`, marks the end of the process of finding the `x`th character. In this example, the 1st character has been determined to be ñdò ï the ascii character with code 100. The search for the 2nd character starts next, but is not completed in Fig. 2.

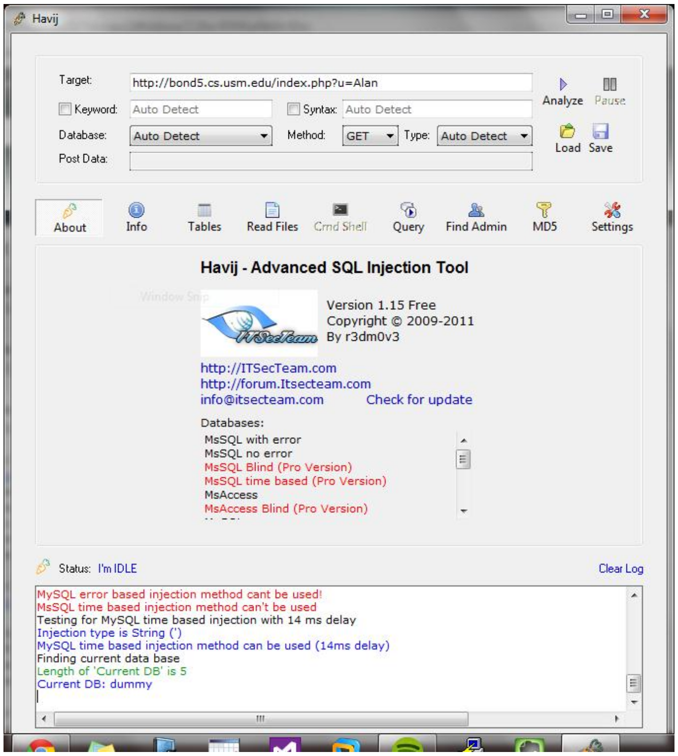


```
GET /?u=alan and if(1=1,BENCHMARK(26000,MD5(0x41)),0)

GET /?u=alan' and if(1=1,BENCHMARK(26000,MD5(0x41)),0) and 'x'='x

GET /?u=alan' and if(Length((database()))<32,BENCHMARK(32716,MD5(0x41)),0) and 'x'='x

GET /?u=alan' and if(Length((database()))<16,BENCHMARK(32716,MD5(0x41)),0) and 'x'='x

GET /?u=alan' and if(Length((database()))<8,BENCHMARK(32716,MD5(0x41)),0) and 'x'='x

GET /?u=alan' and if(Length((database()))<4,BENCHMARK(32716,MD5(0x41)),0) and 'x'='x

GET /?u=alan' and if(Length((database()))<6,BENCHMARK(32716,MD5(0x41)),0) and 'x'='x

GET /?u=alan' and if(Length((database()))=5,BENCHMARK(32716,MD5(0x41)),0) and 'x'='x

GET /?u=alan' and if(ascii(substring((database()),1,1))<79,BENCHMARK(32716,MD5(0x41)),0) and 'x'='x

GET /?u=alan' and if(ascii(substring((database()),1,1))<103,BENCHMARK(32716,MD5(0x41)),0) and 'x'='x

GET /?u=alan' and if(ascii(substring((database()),1,1))<91,BENCHMARK(32716,MD5(0x41)),0) and 'x'='x

GET /?u=alan' and if(ascii(substring((database()),1,1))<97,BENCHMARK(32716,MD5(0x41)),0) and 'x'='x

GET /?u=alan' and if(ascii(substring((database()),1,1))<100,BENCHMARK(32716,MD5(0x41)),0) and 'x'='x

GET /?u=alan' and if(ascii(substring((database()),1,1))=102,BENCHMARK(32716,MD5(0x41)),0) and 'x'='x

GET /?u=alan' and if(ascii(substring((database()),1,1))=101,BENCHMARK(32716,MD5(0x41)),0) and 'x'='x

GET /?u=alan' and if(ascii(substring((database()),1,1))=100,BENCHMARK(32716,MD5(0x41)),0) and 'x'='x

GET /?u=alan' and if(ascii(substring((database()),2,1))<79,BENCHMARK(32716,MD5(0x41)),0) and 'x'='x

GET /?u=alan' and if(ascii(substring((database()),2,1))<103,BENCHMARK(32716,MD5(0x41)),0) and 'x'='x
```

Fig. 2. Sample of (partial) tcpdump of a Havij attack formatted to be readable

Sometimes, perhaps due to delays in processing by the database, Havij receives non-zero runtimes for false statements that cause the binary search to go out of range or return a wrong length or character. This is usually inconsequential, as the search may be run again and comparing two searches allows the operator to fill in missing or wrong characters. In order to describe the database, Havij first runs these searches for length of the database name. Next it will perform binary search for that number of characters to determine the database name. Once it has the database name it can issue statements to determine the number of tables in the database. From there it will find each table name in a similar manner to the way it finds the database name, targeting the length of table names first and then each character of the table names. It may then do this for column names in each table, and then for data contained in the table. Once the structure of the database is known, a hacker may execute arbitrary commands by filling in the appropriate values.

Fig. 3 shows an attack where Havij has determined the length of the databaseôs name to be 5, and then conducted 5 separate binary searches for the characters in the databaseôs name, discovering the name ñdummyò. Fig. 4 shows an advanced stage of this attack where Havij has discovered the name of a table (ñusersò) in the database ñdummyò, and used it to discover the three field names ñuserò, ñemailò and ñpasswordò. This attack can be manually continued through Havij, by selecting the columns in Fig. 4, and clicking ñGetDataò, to reveal the contents of the table, potentially compromising sensitive data.



Target: http://bond5.cs.usm.edu/index.php?u=Alan
□ Keyword: Auto Detect
□ Syntax: Auto Detect
Database: Auto Detect
Method: GET
Type: Auto Detect
Post Data:

Havij - Advanced SQL Injection Tool

Version 1.15 Free
Copyright © 2009-2011
By r3dm0v3

http://ITSecTeam.com
http://forum.Itsecteam.com
info@itsecteam.com Check for update

Databases:
MsSQL with error
MsSQL no error
MsSQL Blind (Pro Version)
MsSQL time based (Pro Version)
MsAccess
MsAccess Blind (Pro Version)

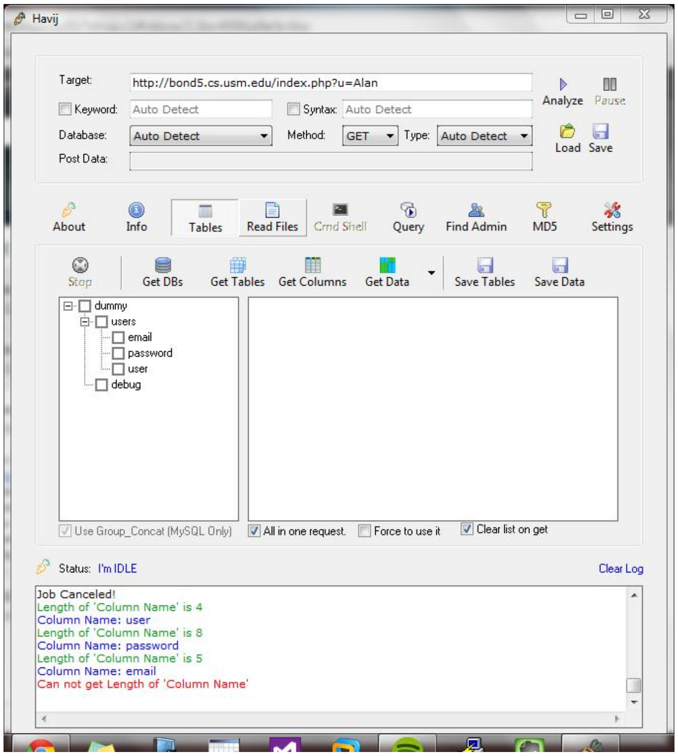
Status: I'm IDLE

```
MySQL error based injection method cant be used!
MsSQL time based injection method can't be used
Testing for MySQL time based injection with 14 ms delay
Injection type is String (')
MySQL time based injection method can be used (14ms delay)
Finding current data base
Length of 'Current DB' is 5
Current DB: dummy
```

Fig. 3. Havij after finding the name of our database ñdummy.ò

### C. Snort

We decided to use the popular packet sniffer Snort to detect these attacks [8]. Snort compares the content of packets against a library of rules, and upon finding a packet whose contents match a rule, may raise an alert, log the packet, drop the packet, or perform some user defined function. With appropriate rules, Snort easily detects the Havij attack, but the functionality of Snort is greatly diminished by the large volume of alerts it raises. For example: our Snort rule library checks the packet content for the ñBENCHMARKò command that Havij issues to check the results of its binary search. This causes Snort to alert hundreds of times for one Havij attack. This problem is worse if valid traffic can contain suspicious content. For example, the character ñ'ò is often needed in SQL injection commands to end the query that is intended to run and allow the arbitrary commands to be inserted, but ñ'ò may also be part of valid names like ñO'Reillyò. A snort rule that checks for ñ'ò in the packet will alert on the name ñO'Reillyò unless additional conditions are added to the rule. Each additional condition to reduce false positives makes the rule easier to defeat. This results in a tradeoff between reducing false positives and decreasing detection rate. Since Snort only considers one packet at a time, it is very difficult to avoid false positives. This is where SIEMs come in.



Target: http://bond5.cs.usm.edu/index.php?u=Alan
□ Keyword: Auto Detect
□ Syntax: Auto Detect
Database: Auto Detect
Method: GET
Type: Auto Detect
Post Data:

```
⊟ □ dummy
   ⊟ □ users
        □ email
        □ password
        □ user
     □ debug
```

☑ Use Group_Concat (MySQL Only) ☑ All in one request. □ Force to use it ☑ Clear list on get

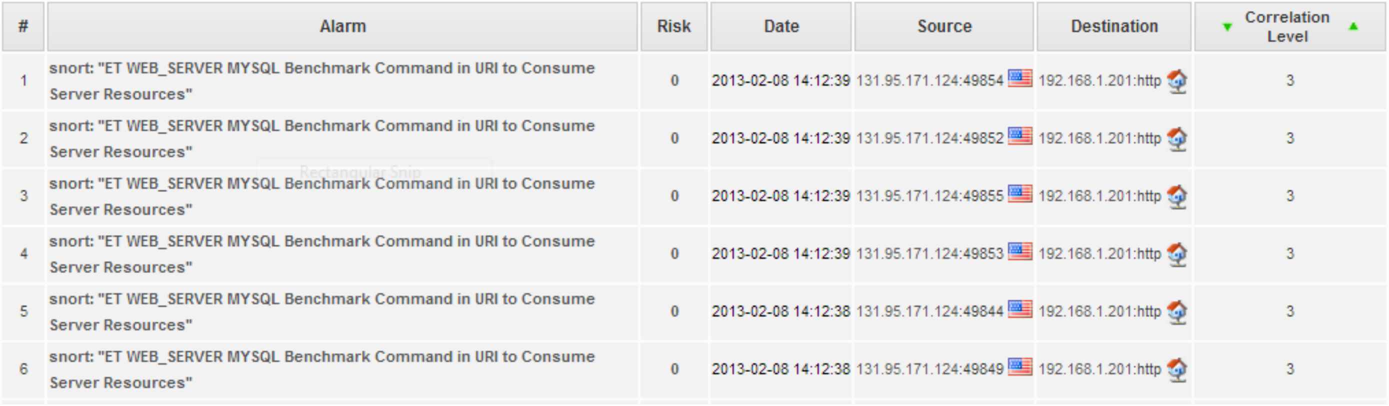
Status: I'm IDLE

```
Job Canceled!
Length of 'Column Name' is 4
Column Name: user
Length of 'Column Name' is 8
Column Name: password
Length of 'Column Name' is 5
Column Name: email
Can not get Length of 'Column Name'
```

Fig. 4. Havij after identifying each column in the users table.

### D. SIEM/OSSIM

SIEM stands for Security Information and Event Management. A SIEM uses tools like snort to detect various low level events, but interprets the results at a higher level before making alerts to the operator. Companies are increasingly using SIEM solutions to meet regulations and increase security [7]. We used the open source SIEM OSSIM for this project. OSSIM uses what its creators call a *correlation engine* to reduce false positives. The correlation engine relies on user created correlation directives to determine when to raise an alert. A correlation directive takes data from one or more sensors, like Snort, and tries to match them to patterns of malicious activity by organizing the data into correlation levels. The first level is always a single occurrence of a suspicious activity. Instead of alerting the operator immediately, the correlation directive moves to level two which will have a set of conditions and a timeout. If the conditions of level two are met before the timeout, the directive will elevate to level three and begin trying to meet a new set of conditions with a new timeout. The user defines how reliable

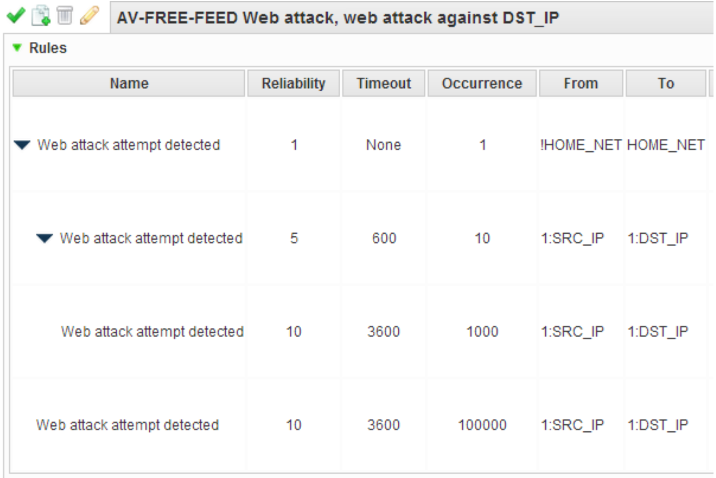


| # | Alarm | Risk | Date | Source | Destination | Correlation Level |
|---|---|---|---|---|---|---|
| 1 | snort: "ET WEB_SERVER MYSQL Benchmark Command in URI to Consume Server Resources" | 0 | 2013-02-08 14:12:39 | 131.95.171.124:49854 | 192.168.1.201:http | 3 |
| 2 | snort: "ET WEB_SERVER MYSQL Benchmark Command in URI to Consume Server Resources" | 0 | 2013-02-08 14:12:39 | 131.95.171.124:49852 | 192.168.1.201:http | 3 |
| 3 | snort: "ET WEB_SERVER MYSQL Benchmark Command in URI to Consume Server Resources" | 0 | 2013-02-08 14:12:39 | 131.95.171.124:49855 | 192.168.1.201:http | 3 |
| 4 | snort: "ET WEB_SERVER MYSQL Benchmark Command in URI to Consume Server Resources" | 0 | 2013-02-08 14:12:39 | 131.95.171.124:49853 | 192.168.1.201:http | 3 |
| 5 | snort: "ET WEB_SERVER MYSQL Benchmark Command in URI to Consume Server Resources" | 0 | 2013-02-08 14:12:38 | 131.95.171.124:49844 | 192.168.1.201:http | 3 |
| 6 | snort: "ET WEB_SERVER MYSQL Benchmark Command in URI to Consume Server Resources" | 0 | 2013-02-08 14:12:38 | 131.95.171.124:49849 | 192.168.1.201:http | 3 |

Fig. 5. Snort activations and correlation level 3

each level is in indicating an attack, and this value along with the user assigned value of the assets that the SIEM is monitoring determines when an alert is actually raised. While the Havij attack generates hundreds of lower levels alerts, the correlation engine raises only one alarm. Fig. 5 shows the directive accumulating multiple snort activations while the far right column displays the correlation level of 3 where the single alert is raised.

This directive generates an alert at level three. It is activated by Snort detecting the `BENCHMARK` command in the packet content. Upon initial detection of the command and elevation to level two, it looks for fifty activations in 6 seconds between the same source and destination IPs that activated level one. If it sees fifty activations before the timeout, an alert will be raised and it will elevate to level 3 where it attempts to collect 1000 activations in the next 10 seconds between the same source and destination IPs. This directive easily picks up on a Havij attack, which generates hundreds of `BENCHMARK` commands within a few seconds in order to perform the binary searches. The details of the directive appear in Fig. 6.



AV-FREE-FEED Web attack, web attack against DST_IP

Rules

| Name | Reliability | Timeout | Occurrence | From | To |
|---|---|---|---|---|---|
| Web attack attempt detected | 1 | None | 1 | !HOME_NET | HOME_NET |
| Web attack attempt detected | 5 | 600 | 10 | 1:SRC_IP | 1:DST_IP |
| Web attack attempt detected | 10 | 3600 | 1000 | 1:SRC_IP | 1:DST_IP |
| Web attack attempt detected | 10 | 3600 | 100000 | 1:SRC_IP | 1:DST_IP |

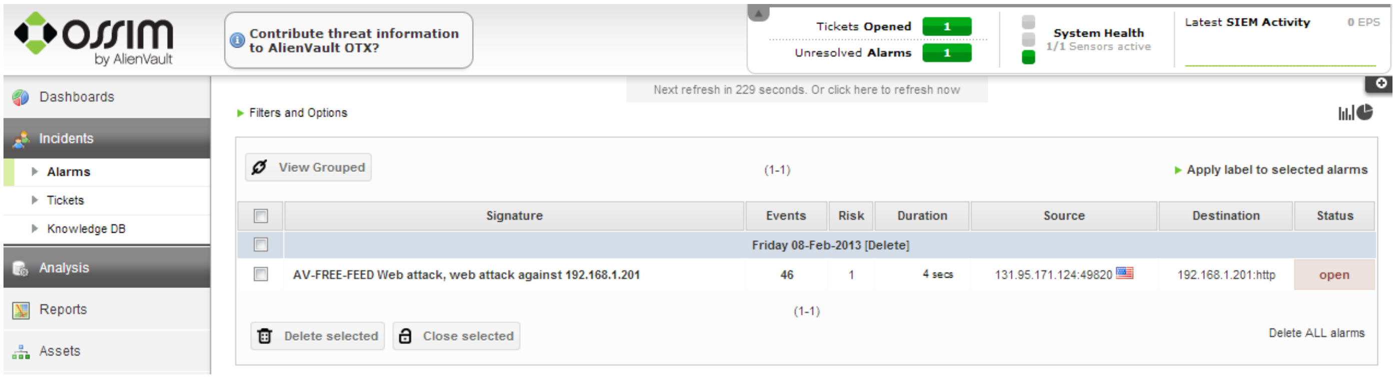
Fig. 6. The OSSIM correlation directive fired upon Havij attack

### *E. Simulating a Directive*

Unfortunately, although OSSIM worked expectedly with the default conditions of a Havij attack, we had difficulty getting it to perform consistently in a distributed attack scenario. This has more to do with the setup of OSSIM, rather than its operation. Due to limited resources and time, we chose to simulate the directive of Fig. 6 with a python script and a tcpdump file. Tcpdump is a utility that captures traffic across a network in a widely used format. We used tcpdump to capture traffic from an attack. We then used a script to create a log of all the packets that contained the `BENCHMARK` command to simulate the snort activations. Using this data, our script counted up the number of activations before the timeout for each IP, elevating to level three in the same way that the OSSIM correlation would. An alert was then raised if enough activations were found. This is shown in the top part of Fig. 8, where 150 events were detected within 6 seconds leading to the outcome ñ`Attack Confirmed`ò.

Next, to simulate a distributed attack, we modified the IP addresses in the attack traffic so that after every 20 packets the IP would change, and these changes cycle within a set of 6 distinct IP addresses. This is a realistic simulation of a distributed attack, especially in a cloud computing environment, where a user can launch multiple instances with distinct IP addresses. By contrast, multiple VMs on a single machine do not acquire distinct external IP addresses (but they do acquire distinct internal addresses). After distributing the attack across the 6 distinct IP addresses, the script was still able to detect each attack, but since activations for any single IP address never exceeded the conditions, each distinct source IP remained at level two and raised no alarm. This is shown in the middle part of Fig. 8, where each source IP address was exonerated as ñ`False Alarm`ò. However, the total number of packets sent by any single IP address is not under 50 (as shown in the bottom part of Fig. 8 for the single source `137.24.100.201`), indicating that it is the temporal staggering of the packets that defeats level 2 of the directive, not a straightforward distribution of the packets among multiple sources which would make each source count fall under the threshold of 50 packets.

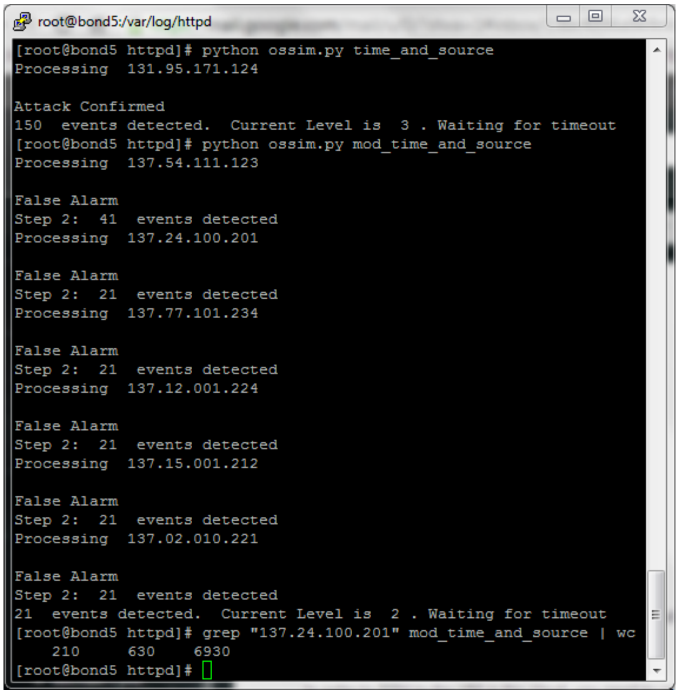
While a straightforward remedy to the above distributed attack scenario is to ignore the source IP addresses (which



| □ | Signature | Events | Risk | Duration | Source | Destination | Status |
|---|---|---|---|---|---|---|---|
| □ | Friday 08-Feb-2013 [Delete] | | | | | | |
| □ | AV-FREE-FEED Web attack, web attack against 192.168.1.201 | 46 | 1 | 4 secs | 131.95.171.124:49820 | 192.168.1.201:http | open |

Fig. 7. The OSSIM correlation directive shown in Fig. 6 is the only incident/alarm that OSSIM fires upon Havij attack.

entails a simple modification of the directive of Fig. 6), it is conceivable that several independent nominal users could actually be performing a similar pattern of activities, raising the possibility of false alarms. In general, for any directive expecting $x$ activations within time $t$ before raising an alarm, $n$ activations could be spread over more than $(n/x)$ IP addresses such that an IP address is not reused before $t$, where lowering $x$ makes it harder to slip past but more likely to raise false alarms. We propose to delve deeper into the (distributed) activities themselves, than most IDSs do, to detect a malicious pattern.



```
[root@bond5 httpd]# python ossim.py time_and_source
Processing  131.95.171.124

Attack Confirmed
150  events detected.  Current Level is  3 . Waiting for timeout
[root@bond5 httpd]# python ossim.py mod_time_and_source
Processing  137.54.111.123

False Alarm
Step 2:  41  events detected
Processing  137.24.100.201

False Alarm
Step 2:  21  events detected
Processing  137.77.101.234

False Alarm
Step 2:  21  events detected
Processing  137.12.001.224

False Alarm
Step 2:  21  events detected
Processing  137.15.001.212

False Alarm
Step 2:  21  events detected
Processing  137.02.010.221

False Alarm
Step 2:  21  events detected
21  events detected.  Current Level is  2 . Waiting for timeout
[root@bond5 httpd]# grep "137.24.100.201" mod_time_and_source | wc
    210     630    6930
[root@bond5 httpd]# 
```

Fig. 8. Top: The attack from a single IP source, that raises an alert from the correlation angine. Middle: Attack spread across 6 source IP addresses. Events are detected but level 2 is not passed for any source, so no alert is raised by the correlation engine. Bottom: A single IP source sends more than 50 packets (210 packets) in all, showing that level 2 was defeated by temporal staggering of the packets.

## IV. Multi-Agent Plan Recognition

Multi-agent plan recognition (MAPR) refers to the problem of explaining the observed behavior trace of multiple agents by identifying the (dynamic) team-structures and the team plans (based on a given plan library) being executed, as well as predicting their future behavior [1][2].

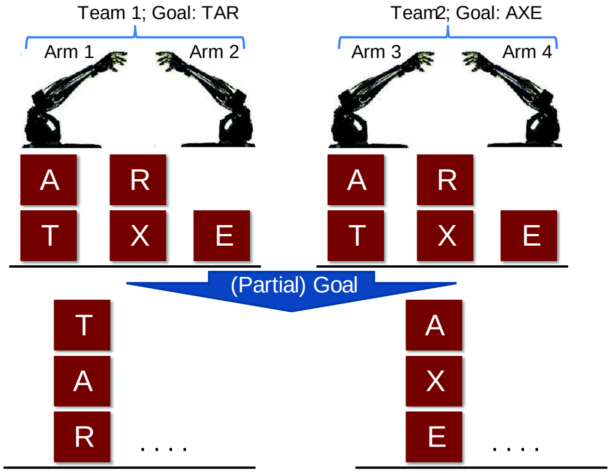
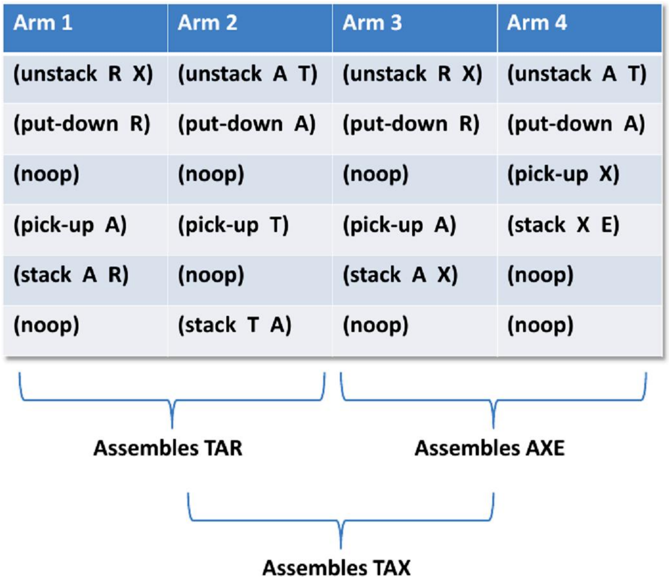


Fig. 9. Multi-agent blocks world example.

We first illustrate MAPR in a multi-agent blocks word domain, shown in Fig. 9, Fig. 10, and Fig. 11, using standard PDDL operators. In Fig. 9 we see two teams of robotic arms assemble (i.e., spell out) the goal words ``TAR" and ``AXE" from separate stacks, starting from the (not necessarily) same initial configuration. Fig. 10 shows the trace of 6 steps of activities of the 4 robotic arms available to the (remote) recognizer, who is not aware of the team-structure (i.e., the mapping of agent-id to stack-id). This assumption partly models the realistic incomplete information under which the recognizer must operate. While arms 1 and 2 appear to jointly assemble ``TAR", and arms 3 and 4 appear to jointly assemble ``AXE", arms 2 and 3 seem to assemble ``TAX" as well, creating ambiguity for the recognizer. The key insight is to *partition* the trace into non-overlapping team plans, such that invalid teams (such as the supposed team of agents 2 and 3) fail to yield a complete partition hypothesis. In this example, agents 1 and 4 would be executing illegal plans individually,



| Arm 1 | Arm 2 | Arm 3 | Arm 4 |
|---|---|---|---|
| (unstack R X) | (unstack A T) | (unstack R X) | (unstack A T) |
| (put-down R) | (put-down A) | (put-down R) | (put-down A) |
| (noop) | (noop) | (noop) | (pick-up X) |
| (pick-up A) | (pick-up T) | (pick-up A) | (stack X E) |
| (stack A R) | (noop) | (stack A X) | (noop) |
| (noop) | (stack T A) | (noop) | (noop) |

Assembles TAR
Assembles AXE
Assembles TAX

Fig. 10. Trace of activities of 4 robotic arms, shown in Fig. 9

or building separate stacks as a team, neither of which yields a valid partition hypothesis. Fig. 11 shows a (non-unique) plan from the library, for start state in Fig. 9 and goal ``TAR'', in the form of a plan graph. This is a graph based on the partially ordered set of steps needed to achieve a goal from a start state, with added constraints for multi-agency: *role constraints* (which steps need to be performed by the same agent) and *concurrency constraints* (which steps need to be executed simultaneously; not needed in this illustration). The above illustration is adopted from a previous paper by the authors[2].

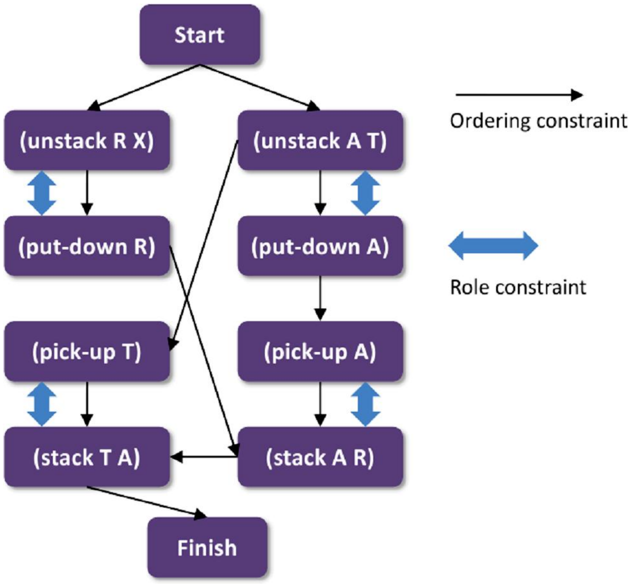


Fig. 11. A plan graph for the blocks world example.

## V. Application of MAPR for Detection of Havij Attack

The SQL injection attack of Havij follows a pattern that can yield the abstract plan graph shown in Fig. 12.

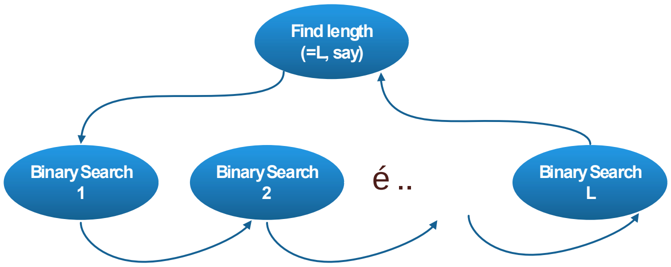


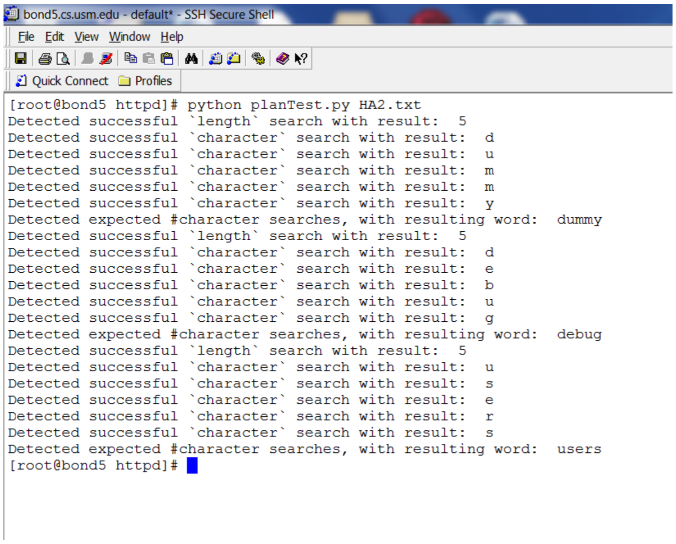
Fig. 12. Abstract plan graph corresponding to Havij attack

Here a binary search first finds the length of a certain field, say L. Then L binary searches are done in succession, followed by a return to the top (abstract) action. Suppose the *i*th binary search returns a character that is used to fill the *i*th character of a string *s*. Then the string *s*[1:L] will be a part of the query used in the next search, e.g., after the name of a database is found this way, the queries to detect the names of tables in that database will include the name of the database already found. A string that differs by only a few characters from the database name used later should still be accepted because of the occasional false positives in the `BENCHMARK` command. This pattern repeats to find the names of tables in the database using the database name, and then again to find the column names in a table using that tableôs name.

A solution to the problem of limiting false positives while still detecting an attack that is spread across multiple agents is to use plan recognition. When a user switches IP addresses to appear as multiple users, the observable effect is indeed equivalent to there being multiple users. Hence, we can indeed treat this as a *multi-agent* plan recognition problem. Rather than relying on a single IP generating sufficient suspicious activity to raise an alert, a plan recognition algorithm searches input from all users to see if steps in a plan have been completed. We have developed and tested a simple Python script that finds the length command and then identifies the search result by finding the last ñ=ò or equivalent symbol. It then looks for that number of binary searches using the `ascii` and `substring` commands. If it sees these actions it is reasonable to assume that a field name has been found whether spread across multiple agents or not.

We have tested our script on the same tcpdump file that was used with our simulation of OSSIMôs correlation directive, where we showed successful camouflage of a Havij attack before. Fig. 13 shows the output of this script, not only detecting the attack despite distributed sources, but also revealing what the attacker has learned, viz., the words ñ`dummy`ò, ñ`debug`ò and ñ`users`ò that are names of actual entities in the database (see Fig. 4). Thus the ideas underlying

MAPR enable us to detect a SQL injection attack where a traditional IDS might fail in a cloud computing environment.


bond5.cs.usm.edu - default* - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles

```
[root@bond5 httpd]# python planTest.py HA2.txt
Detected successful `length` search with result:  5
Detected successful `character` search with result:  d
Detected successful `character` search with result:  u
Detected successful `character` search with result:  m
Detected successful `character` search with result:  m
Detected successful `character` search with result:  y
Detected expected #character searches, with resulting word:  dummy
Detected successful `length` search with result:  5
Detected successful `character` search with result:  d
Detected successful `character` search with result:  e
Detected successful `character` search with result:  b
Detected successful `character` search with result:  u
Detected successful `character` search with result:  g
Detected expected #character searches, with resulting word:  debug
Detected successful `length` search with result:  5
Detected successful `character` search with result:  u
Detected successful `character` search with result:  s
Detected successful `character` search with result:  e
Detected successful `character` search with result:  r
Detected successful `character` search with result:  s
Detected expected #character searches, with resulting word:  users
[root@bond5 httpd]#
```

Fig. 13. Multi-agent plan recognition successfully detects Havij plan of Fig. 12 in the tcpdump, showing what the attacker has learned.

## VI. CONCLUSIONS

In this paper we have argued that cloud computing environments offer opportunities for cyber-attackers to distribute their attacks, in particular SQL injection attacks, so as to defeat a traditional intrusion detection system. In such a scenario, we have demonstrated that the application of multi-agent plan recognition enables us to successfully detect a distributed SQL injection attack. Our deployment of MAPR toward cloud-computing security is currently limited to Havij-style SQL injection attacks only. However, the purpose of this paper is to demonstrate the feasibility of such a deployment in a realistic (instead of an abstract) scenario, rather than building a comprehensive suite of cyber-attack detection mechanisms. The latter is, in fact, a long term goal of this research.

A major limitation of the MAPR approach in detecting attacks is that the attack must follow a plan with multiple distinct steps. A brute force attack, for example, repeats the same action of attempting to log in many times until a login is guessed. This would be better handled by the correlation directives already in place. However, when there is a distinct plan it becomes easier to determine the validity of the attack, how successful the attacker was, and what the attacker is likely to do next by using the MAPR approach. Our experiment is also limited by the fact that our testing was performed against a simulated version of the OSSIM directive, which should perform the same way but this cannot be fully guaranteed. Also, running our test against log files does not test whether this method of detection is practical for examining larger volumes of traffic in real time.

In the future we would like to explore other types of increasingly complex cyber-attacks in the context of cloud computing and apply MAPR and evaluate its effectiveness in detecting such attacks. Even SQL injection attacks can be done in other (non-Havij) ways. Therefore our goal would require building (either manually or data mined from labeled data) a plan library for different variants of each kind of attack, for a range of different attack types. We plan to leverage our current setup to easily collect the required data to incrementally build a more comprehensive plan library.

## VII. ACKNOWLEDGMENT

The authors thank the anonymous reviewers for feedback, and the NASA Office of Chief Technologist at NASA Stennis Space Center for support under the 2012 Center Innovation Fund.

## REFERENCES

[1] B. Banerjee, L. Kraemer, and J. Lyle. Multi-Agent Plan Recognition: Formalization and Algorithms. In Proceedings of AAAI-10, pp. 1059ï 1064, Atlanta, GA, 2010.

[2] B. Banerjee and L. Kraemer. Branch and Price for Multi-Agent Plan Recognition. In Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-11), pp. 601ï 607, San Francisco, CA, 2011.

[3] Halfond, W. G., Jeremy Viegas, and Alessandro Orso. "A classification of SQL-injection attacks and countermeasures." In Proceedings of the IEEE International Symposium on Secure Software Engineering, pp. 65-81. IEEE, 2006.

[4] Hankz. H. Zhuo and Lei Li. Multi-agent plan recognition with partial team traces and plan libraries. In Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11), pages 484ï 489, 2011.

[5] Hankz. H. Zhuo, Qiang Yang, and Subbarao Kambhampati. Action-model based multi-agent plan recognition. In Proceedings of NIPS 2012, 2012.

[6] Karg, . OSSIM, "Correlation engine explained.." Last modified 2004/02/01. Accessed March 5, 2013. http://www.alienvault.com/docs/correlation_engine_explained_rpc_dcom_example.pdf.

[7] Nicolett, Mark, and Kelly M. Kavanagh. "Magic Quadrant for Security Information and Event Management." Gartner RAS Core Reasearch Note (May 2009) (2011).

[8] Roesch, Martin. "Snort-lightweight intrusion detection for networks." In Proceedings of the 13th USENIX conference on System administration, pp. 229-238. 1999.

[9] Zafarullah, Z.; Anwar, F.; Anwar, Z., "Digital Forensics for Eucalyptus," Frontiers of Information Technology (FIT), 2011 , vol., no., pp.110,116, 19-21 Dec. 2011 doi: 10.1109/FIT.2011.28

[10] Nurmi, Daniel, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. "The eucalyptus open-source cloud-computing system." In Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on, pp. 124-131. IEEE, 2009.

[11] Karg, D., and J. Casal. Ossim: Open source security information management. Tech. report, OSSIM, 2008.