IMPLEMENTATION OF HONEYTOKEN MODULE IN DBMS ORACLE 9iR2 ENTERPRISE EDITION FOR INTERNAL MALICIOUS ACTIVITY DETECTION

Antanas Čenys[1,3], Darius Rainys[1,2], Lukas Radvilavičius[1,3], Nikolaj Goranin[4]

[1]Informtion Systems Laboratory, Semiconductor Physics Institute
A.Goštauto11, LT-01108, Vilnius, Lithuania
cenys@uj.pfi.lt
[2]UAB "BlueBridge",
Jasinskio. 16, LT-01112, Vilnius, Lithuania
darius.rainys@bluebridge.lt
[3,4] Vilnius Gediminas Technical University,
Saulėtekio 11, LT-10223, Vilnius, Lithuania
[3]lukas@fmf.vtu.lt [4] ngrnn@fmf.vtu.lt

SUMMARY

Information stored in companies' databases is often the most valuable one and as a result database security is of great importance. It is a very complex task, however, to ensure database security without limiting capabilities and productivity of legal users. Honeytokens are one of the new methods to increase information security. The method uses fake information resources to attract illegal users and identify them. In the paper implementation of honeytoken module for Oracle 9iR2 database management system is described. Three original modules were programmed to combine Oracle Fine-Graned auditing features, internal triggers and functions reporting on illegal access to fake resource.

INTRODUCTION

Data and information are sometimes called the "crown jewels" of an organization **[1]**. Modern company cannot exist without flexible, fast and secure data management system. Without such a system based on relational or any other up-to-date (object-oriented, object-relational) DBMS company is not competitive on the market. Insuring data security is of vital importance for any company due to many different reasons: stolen sensible data can be used by competitors, information leakage can discredit company's name or even lead to bankruptcy, etc.

To ensure data security various methods and tools are used. According to cyber-security specialist Bruce Schneier's definition given in his book, "Secrets and Lies"**[2]**, security could break into three areas: prevention, detection, and reaction. Nowadays most of the companies use passive data protection mechanisms, i.e. firewalls, IDS and NIDS, and security audits. Internal and external audits can be used to detect "weak" places in company's security policy and its implementation. Audits also can detect past

successful attacks, if hacker didn't clean the logs. Firewall and IDS solutions are effective in detecting and preventing simple and/or well-known illegal access methods. Network level firewalls (generation I) usually use pre-defined rules blocking access to specific network ports from certain IP numbers or ranges and they correspond to a classical prevention method. IDS and generation II (program level) firewalls can detect, block and alarm on known exploits and attack mechanisms by the use of attack "signatures". These tools are quite effective for detection purposes. But on the other hand IDS have serious weaknesses such as a large number of false alarms, huge and difficult to maintain reports, disability to detect new attacks with unknown signature or situation when medium/high-skilled attacker is able to modify slightly the signature of a well-known attack. Anomaly analysis based IDS can at least partially help solving these problems.

All above security tools and methods are passive by definition. More pro-active method is to prepare a special "traps" for an attacker known as honeypots. The first articles on honeypots were published in late 80-ies, early 90-ies **[3-4]**. A honeypot is defined as an information system resource whose only value lies in unauthorized or illicit use of that resource **[5].** It can be used in different ways for prevention, detection and reaction. Different aspects of honeypot usage were described in **[5-6]** and "Honeypot Alliance" white papers. Up to now, however, there are few solutions or reported research on honeypots usage for DBMS protection. Traditionally honeypots for DBMS protection are described as real DBMS servers or DBMS server emulators installed as a part of honeynet – network of honeypots. This system can be used for detection and reaction to external attacks but not against malicious employees or clients having legal access to the database. "Classical" honeypot cannot help to detect legal database user trying to obtain information for which he does not have access permission. On the other hand detecting illegal activities of legal users is very important since according to recent statistics up to 80% of cyber-crimes are commited by employees. One of the solutions tackling this problem can be deployment of specific 'traps" – so called honeytokens. The term honeytoken was first coined by Augusto Paes de Barros in 2003 on the honeypots mailing list. A honeytoken like a honeypot is attractive for attacker information resource without any legal use. Any interaction with a honeytoken most likely represents unauthorized or malicious activity **[7]**. Any data resource – DB, DB table, file or letter can be used as honeytoken. Lance Spitzner's whitepaper on honeytoken got more feedback than any other on honeypots before. The reasons for such interest are simplicity of honeytokens, low cost and effectiveness in detecting internal malicious activity. Honeytokens are not designed specifically to detect attackers or prevent attacks, but are a flexible and simple tool with multiple security applications. Pete Herzog, managing director of the Institute for Security and Open Methodologies, says that he has used honeytokens to detect when employees illicitly download forbidden material. For example, he has entered corporate memos with particular typos into

private databases and then monitored company networks to see where those typos show up. Tracing these honeytokens, he says, often leads to catches of illegal materials stored on the network. **[8]**.

In his article Mr. Lance Spitzner suggests one concept of honeytoken implementation in database. "*For example, the credit card number 4356974837584710 could be embedded into database, file server, or some other type of repository. The number is unique enough that there will be minimal, if any, false positives. An IDS tool, such as Snort, could be used to detect when that honeytoken is accessed. Such a simple signature could look as follows*:

```
alert ip any any -> any any (msg:"Honeytoken Access - Potential unauthorized Activity";
content:"4356974837584710";)
```

It was the only publicly available solution of honeytoken usage for DBMS security we were able to find. Spitzner's idea is attractive due to its universality and versatility. However such simple version of honeytoken has some disadvantages: it won't work if SQL query output is provided in encrypted form, additional computer system is needed since running sniffer on the same machine as DBMS may look suspicious for hacker, DBMS log analysis for access to honeytoken may be misleading, since logs might be modified or polluted.

Presented in the paper concept for honeytoken implementation is platform specific (runs on DBMS Oracle 9i EE or higher), but does not possess disadvantages described above. It is based on Oracle FGA mechanism with some additional original modules. It was first described in **[9]** and later installed, tested and improved at Vilnius Gediminas Technical University computer laboratory Oracle 9i EE DBMS server, used for educational purposes.

2. REQUIREMENTS TO DATABASE LEVEL HONEYTOKEN MODULE

Usually access to the database is organized through third-party applications made for accessing and managing data. When data is accessed in such a way, it is not too difficult for a malicious user to gain database usernames and passwords. After gaining login information of database an attacker can connect to the database through SQL client and try to view the data manually. Honeytoken strategy is to insert a table with "sweet" name able to attract malicious user. Any activity with such a table can be registered and an alert message sent to system administrator.

Particular structure of any honeytoken module depends on the aims and goals of the system. The initial requirements and suggestions on the structure of the honeypot module were presented in **[9]** It was proposed that the module should contain analysis and reaction subsystems. It was also proposed to use the SPIKE software package for connection emulation. Taking into consideration previous experience the following goals for the honeytoken module can be formulated: honeytoken module should be a simple and cheap tool to detect illegal activity at database level coming from the inside and outside of the company without decreasing existing level of security. It would be useful also to provide automatic reacting tools and to be able to log the malicious activity. Implementation of such q module would result in database security improvement. Honeytoken log may be used to analyze data stealing methods and be a base for further improvements and changes.

Above formulated goals as well as the general honeytoken strategy lead to specific requirements for the module:

- To be cheap, easy to implement and administrate the module should not be a dedicated system but an integrated part of DBMS able to monitor any type of database objects. The effort should be taken to ensure that the module produces few false-positives and false-negatives.
- For security reasons implementation of the module should not require DBMS to contain specially unpatched known security holes;
- Module should not look suspicious to the attacker and contain real-looking information. For example database table EMPLOYEES should keep a large number of real names with corresponding unreal salaries and social insurance numbers;
- A care should be taken for the module implementation not to contradict national and international laws on provoking of criminal activities;

In addition database tables security settings containing sensitive information should be audited and list of users, accessing them, should be revised. It is also very important to store module's log files on a separate secure storage, since it is assumed that an attacker may have access not only to the DBMS, but to the operating system as well.

To meet the above requirements the following structure of the module consisting of four parts was chosen:

1. Database object (table, view, etc.) with real-like data and attractive ("sweet") name, for example, TABLE CREDIT_CARDS or VIEW EMPLOYEES_SALARY;

2. Object (function / library), monitoring access (insert, select, etc.) to honeytoken object;
3. Object (function / library) launching external software when monitored honeytoken is being accessed.
4. External software module, responsible for logging, administrator alerting and automatic reacting to incidents (for example blocking access). Testing variant of external software module was programmed to alert administrator by sending him e-mail with security violation descriptions and is based on a simple bash-script and open-source utilities. Additional functionality will be added after long-term honeytoken module tests.

3. HONEYTOKEN MODULE REALIZATION FOR DBMS ORACLE 9I EE

Implementation of honeytoken module for particular DBMS depends on its structure and communication with external processess. DBMS Oracle 9i EE has a possibility to launch external processes using Oracle native process (library) “extproc” shown in Figure 1.

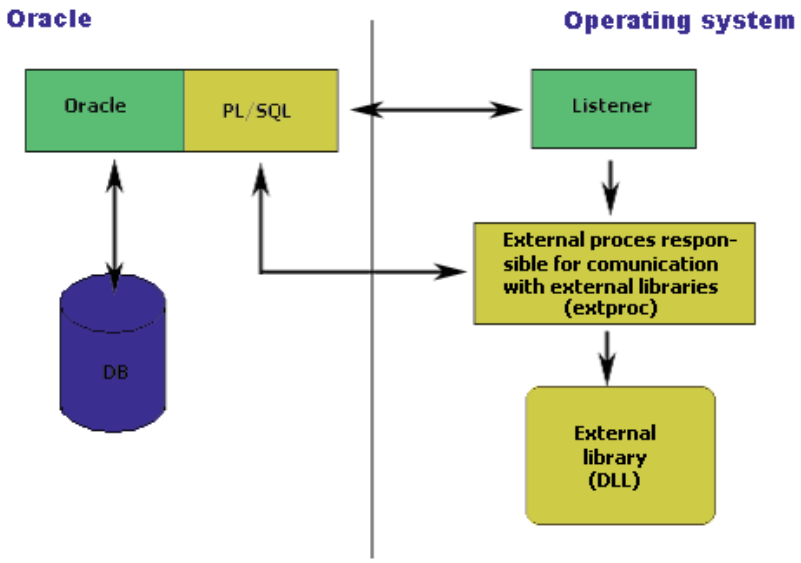

Figure 1. Scheme of DBMS Oracle communications with external processes

“Listener” program is responsible for communication between Oracle DB server and client side applications. It is the only legal way Oracle developers have left for communication with external libraries.

We suggest three possible honeytoken module implementations:

- to use table-simulating pipelined function in combination with the external procedure. This solution does not fulfill all the requirements provided above, so it is described briefly. The main disadvantage is that pipelined function is not listed in the table list and the attacker may not notice it or suspect something wrong.
- to use combination of triggers and external procedure. This solution is the most universal and fulfills the requirements above. The disadvantage of this method is its complexity.
- to use combination of Oracle Fine Grained Auditing (FGA) and external library. This method can be used only for DBMS Oracle versions from 9i EE and higher.

It should be pointed out that some specific DBMS Oracle features influenced realization of the module. Triggers in DBMS Oracle 9i EE can be activated only on “insert” and “update” transactions, but not on “select”. This is a reason why special way-around for second implementation (combination of triggers and external library) is needed. Audit on “sweet” table is turned on. Results are written to a separate audit table. When the audit table is updated, the trigger, activating the external module through a chain of connections is started. In the third FGA case this limitation does not exist, however this implementation can be used only for DBMS Oracle 9i EE and higher.

3.1 Module Insertion Concept

General scheme of module insertion to Oracle 9i EE DBMS is shown in figure 2. Pseudo-connections are shown by the dashed arrows. The provided object links may be not exact, since real internal Oracle DBMS architecture is not in public domain.

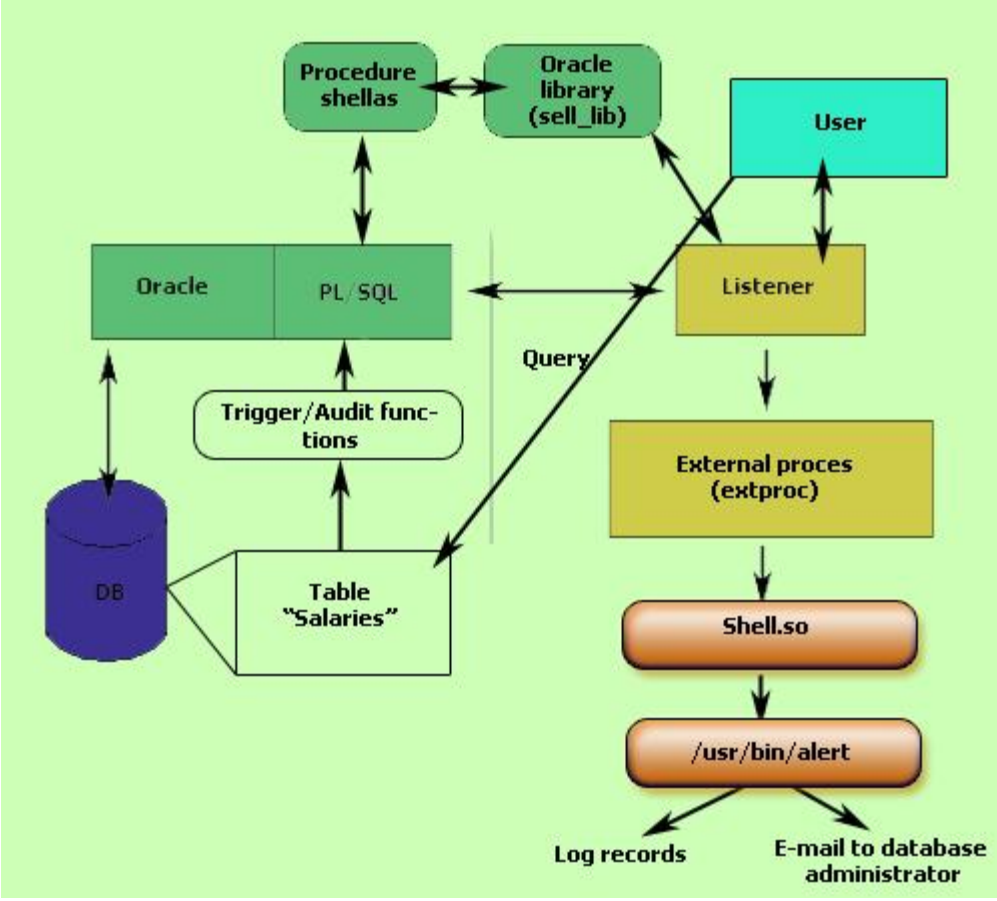

Figure 2. General honeytoken module scheme

To insert a honeytoken module to DBMS a chain of procedures is performed.

1. DBMS Oracle can use specially prepared external libraries, but can not start an external program. So we have to create a library (shell_lib):

```
void sh(char *);
int sh( char *cmd ) {
int num;
num = system(cmd);
return num;
}
```

This library is compiled with internal Oracle libraries as gcc compiler parameters.

DBMS Oracle has a library structure, similar to the Operating system. Because of that we have to create internal Oracle library with all the parameters necessary.

```
CREATE OR REPLACE LIBRARY shell_lib is ‘/opt/oracle/product/9.2.0/lib/shelllib.so';
```

After this action we get shell_lib library. After successful ending we
will get:

```
Library created.
```

2. Oracle configuration files TNSNAMES.ORA and LISTENER.ORA are modified so we could use external procedures.

3. Now we can create an internal Oracle procedure shells. This procedure is called every time the “sweet” table is accessed.

```
CREATE or REPLACE procedure shellas(cmd IN char) as external name "sh" library
shell_lib language C parameters (cmd string);
```

4. External alerting module is programmed. In our case it is a simple bash-script which sends a database administrator notification by e-mail if someone tries to access the monitored table.

```
(GNU bash, version 2.05b.0(1)-release (i686-pc-linux-gnu)).

#!/bin/bash

echo "==============================" >> /var/log/orcl.alerm
echo "Beginning of the record" `date` >> /var/log/orcl.alerm
lsof -i |grep oracle >> /var/log/orcl.alerm
echo "End of the record"
echo `date` >> /tmp/tmp.orc
lsof -i |grep oracle >> /tmp/tmp.orc
cat /tmp/tmp.orc | sendmail admin@herkus.vtu.lt
rm /tmp/tmp.orc
```

The log file generated by this module looks like this:

```
========================================
Beginning of the record Tue Jun 1 18:34:21 EEST 2004
oracle 1032 1758 12u IPv4 3115 UDP localhost:32768
oracle 1032 1758 13u  IPv4 3144 TCP herkus.stml.lan:32770->herkus.stml.lan:1521 (EST…
oracle 1048 1758 12u IPv4 3135 UDP localhost:32769
oracle 1050 1758 12u IPv4 3138 UDP localhost:32770
oracle 1050 1758 13u IPv4 3139 TCP *:32769 (LISTEN)
oracle 21079 1758 14u  IPv4 73200 TCP herkus.stml.lan:1521->212.122.76.172:3645 (EST…
oracle 21081 1758 14u IPv4 73205 TCP herkus.stml.lan:1521->212.122.76.172:3649 (EST…
oracle 21083 1758 14u IPv4 73210 TCP herkus.stml.lan:1521->212.122.76.172:3651
...
oracle 21224 1758 14u IPv4 95660 TCP herkus.stml.lan:1521->b25.vtu.lt:62514 (EST…
oracle 21228 1758 14u IPv4 95669 TCP herkus.stml.lan:1521->212.122.76.172:3369 (EST…
oracle 21230 1758 14u IPv4 95674 TCP herkus.stml.lan:1521->b25.vtu.lt:63189 (EST…
...
End of the record
```

5. Finally we create a honeytoken table and fill it with real-like data:

```
CREATE TABLE Salaries (
Reg_Date Timestamp,
Surname varchar2(20),
Name varchar2(15),
Personal_number varchar2(11));
```

3.2 Pipelined Function Realization

Pipelined function is a special object, which can be used to emulate a database table. In this case triggers are not used, since the pipelined function can access the external process through the sequence of procedures and libraries. The problem is that pipelined function is not listed among database tables. To create the pipelined function one should use such a procedure:

- create object equivalent to the table

```
CREATE OR REPLACE type tabview as object
(Reg_Date Timestamp,
Surname varchar2(20),
Name varchar2(15),
Personal_number varchar2(11));
```

- create tabview table

```
create type tabview_table as table of tabview;
```

- create the function

```
CREATE or REPLACE function Salaries
return tabview_table pipelined as
cursor c1 is
select Reg_Date, Name, Surname, Personal_number
from Salaries;
begin
    for c1_row in c1 loop
        shellas('/usr/bin/aliarmas);
        PIPE ROW   (tabview(c1_row.Reg_Date, c1_row.Name,
                   c1_row.Surname, c1_row.Personal_number));
    end loop;
    return;
end Salaries;
```

Now if the attacker tries to access the “Salaries” table, the corresponding pipelined function will be started and alerting script “aliarmas” will inform the DBMS administrator about the incident.

3.3 Combination of Triggers and External Procedure

Honeytoken+Trigger/External Procedure case is the most universal one and can be used on almost all DBMS Oracle versions. There are only several adjustments you have to do to the main conceptual model to create a honeytoken module of this type.

First of all, you have to create a table, called “Salaries_audit”, where attempts to access the “sweet” table “Salaries” will be logged. The second step is to turn audit on for the “Salaries” table and to create triggers, which are activated when new records are added to the audit table “Salaries_audit”. For example:

```
CREATE trigger danger
  after insert
  on Salaries_audit
  for each row
     begin
     shellas('/usr/bin/alert’);
end danger;
```

Now every time an attacker tries to select, modify or add data to the “Salaries” table the trigger “danger” is activated, as a new record is added to the audit table “Salaries_audit”. It calls procedure “shellas”, which is able to start an external alerting module “alert” with the help of the library “shell_lib”. Alerting module sends an administrator an e-mail and logs all the attacker’s actions to the local file.

3.4 Combination of Oracle Fine Grained Auditing (FGA) and External Library

Traditional Oracle Database auditing options let track the actions users perform on objects at the macro level. For example, auditing SELECT statements on a table it is possible to track who selected data from the table. However it is not possible to detect what is selected. With data-manipulating statements such as INSERT, UPDATE, or DELETE it is possible to capture any changes by using triggers or by using the Oracle LogMiner utility to analyze the archived logs. Because simple SELECT statements are non-data-manipulating, they neither fire a trigger nor go into archived logs that you can mine later, so these two techniques fall short where SELECT statements are concerned.

In Oracle9i Database version a new feature called fine-grained auditing (FGA) was added. This feature allows to audit individual SELECT statements along with exact statements issued by users. In addition to simply tracking statements, FGA provides a way to simulate a trigger for SELECT statements by executing a code whenever a user selects a particular set of data. The power of FGA doesn't stop at merely recording events in audit trails; FGA can also optionally execute procedures. A procedure could perform an action such as sending an e-mail alert to an auditor when a user selects a certain row from a table, or it could write to a different audit trail. This stored code segment, which could be a stand-alone procedure or a procedure within a package, is known as the handler module for a policy **[10].**

To turn the FGA on:

```
execute dbms_fga.add_policy
(object_schema=>'employee',
object_name=>Salaries,
policy_name=>'Salaries_ACCESS',
handler_schema=>'sys',
handler_module=>'shellas');
```

The handler_module parameter defines the procedure, which is called when the monitored table is accessed. The specified internal procedure “shellas” is able to call an external alerting module. It is very comfortable because no additional trigger/audit table combination is needed. It is possible get the audit results by SQL-querying audit tables.

```
SELECT timestamp,
db_user,
os_user,
object_schema,
object_name,
sql_text
FROM dba_fga_audit_trail
WHERE object_name = ‘Altyginimai’;
```

Reaction mechanism can be organized similarly to the triggers/external procedure case.

4. CONCLUSIONS

Possibilities to use honeytokens for DBMS security have been studied. We have developed, implemented, and tested three different honeytoken modules for DBMS Oracle 9i Enterprise Edition: a) honeytoken plus pipelined function; b) honeytoken plus trigers/external procedures; c) honeytoken plus FGA; all based on the same concept of module insertion. Implemented modules can monitor “sweet” objects in database, perform simple reaction procedures, such as e-mail sending to the DBMS administrator and log the malicious activity to external file or database table. All modules were created with the help of free software tools, implemented and tested at Vilnius Gediminas Technical University IT Laboratory. The tests performed at the IT laboratory have shown, that module usage does not increase the load on DBMS and Operating system. In the future we are going to perform tests on industrial DBMS and to evaluate system load for heavily used databases. The test results will be used for the module improvements. If the tests are successful we are planning to create an end-user product with easy to use interface for installation and administration. We also expect to implement similar solutions for Microsoft SQL and IBM DB/2 DBMS servers.

For the databases with a large number of records and users honeytoken modules can be very effective tools for early intrusion detection. It should be pointed out, however, that honeytoken type tools having many advantages such as simplicity of the concept and ability to detect local threats by no way can replace other security tools.

REFERENCES:

[1] **P.Finnigan "**Introduction to Simple Oracle Auditing", **http://www.petefinnigan.com/ ,** April 29, 2003

[2] **B.Schneier, J. Wiley,** "Secrets And Lies: Digital Security in a Networked World", John Wiley & Sons, 2000.

[3] **B.Cheswick,** "An evening with Berferd in which a Cracker is Lured, Endured, and Studied", http://www.tracking-hackers.com/papers/berferd.pdf , 1991.

[4] **C.Stoll**, "Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage", Pocket Books, New York, 1990.

[5] **L.Spitzner,** "Honeypots: Tracking Hackers", http://www.spitzner.net/ 2002

[6] **L.Spitzner,** "Honeypots: Definitions and Value of Honeypots", http://www.tracking-hackers.com/papers/honeypots.html, 2003.

[7] **L. Spitzner,** "Honeytokens: The Other Honeypot", http://www.securityfocus.com/infocus /1713, 2003.

[8] **N.Thompson**, "New Economy", April 28, 2003

[9] **A. Čenys, D. Rainys, L. Radvilavičius, A. Bielko** "Development of Honeypot System Emulating Functions of Database Server", NATO conference, "Symposium on Adaptive Defense in Unclassified Networks", France, 2004 April 19-20.

[10] **A.Nanda**, „Fine-Grained Auditing for Real-World Problems", http://www.oracle.com/technology/oramag/webcolumns/2003/techarticles/nanda_fga.html