Bortoli Tomas 181654
Mariotto Alex 183818
Dallatorre Giulio 183816

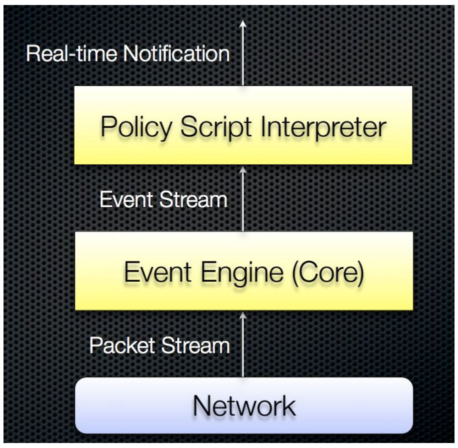
NETWORK SECURITY LAB RELATION - GROUP 3 TOPIC 12

Au Lecteur

This relation cover the work done to prepare and to do the laboratory of network security in the malware lab location. An important note is that to have a full-view of our work, it would make sense to take a look also at the attack and detection scripts residing on our *VMs*.

Bro NIDS Introduction

Our laboratory topic was on *Bro Nids*, a powerful Network Intrusion Detection System; that based on our experience should probably be the most advanced FLOSS (Free and Open Source Software) of this kind currently on the global scene.
Bro's power is over the common packet sniffers like *Wireshark* or *Tcpdump,* because with Bro it's possible to process the content of the packet in a *Turing-Complete* way.
It is much more dynamic and flexible than wireshark's filter, and it is also possible to process the packets offline, providing raw .pcap files.



A simple schematic representation of the software structure and behaviour (on the left).
It shows that the packets coming from the Bro's observed network are labeled and considered as Events (Event-Driven Programming).
This interesting feature provides to the user an interesting platform to work with defined in the Policy Script Interpreter part that is Bro Scripting Language.
It consists in a complete programming language that is C/python-like, so if the user posses good programming skills it will have no problems to fit its needs by implementing its own customised network policies.

This imply that actions taken from the packet analysis may vary from simply logging strange activities, network attacks, or network congestions, to sending emails or sms to network administrator and provide useful informations. It can be also possible to cut-off a host from the network if Bro is running on a master router or switch (it should run there, to sniff all the network traffic).

This happens because from a bro script it is possible to also launch bash commands. Anyway, by default Bro runs without customized scripts and it provides a lot of useful logs (generated by default) collecting the network activity, some these are:

- conn.log
  - TCP connections info
  - UDP connections info
  - ICMP connections info
- dhcp.log
  - DHCP leases
- ssl.log
  - SSL handshake info
  - TLS handshake info
- dns.log
  - DNS Requests
  - DNS Replies
- ftp.log
  - FTP activity
- irc.log
  - IRC Commands
  - IRC Replies
- files.log
  - file transmissions under different protocols
- reporter.log
  - Internal Errors
  - Internal Warnings
  - Info messages
- packet_filter.log
  - Lists packet filters that are applied
- http.log
  - HTTP requests
  - HTTP replies

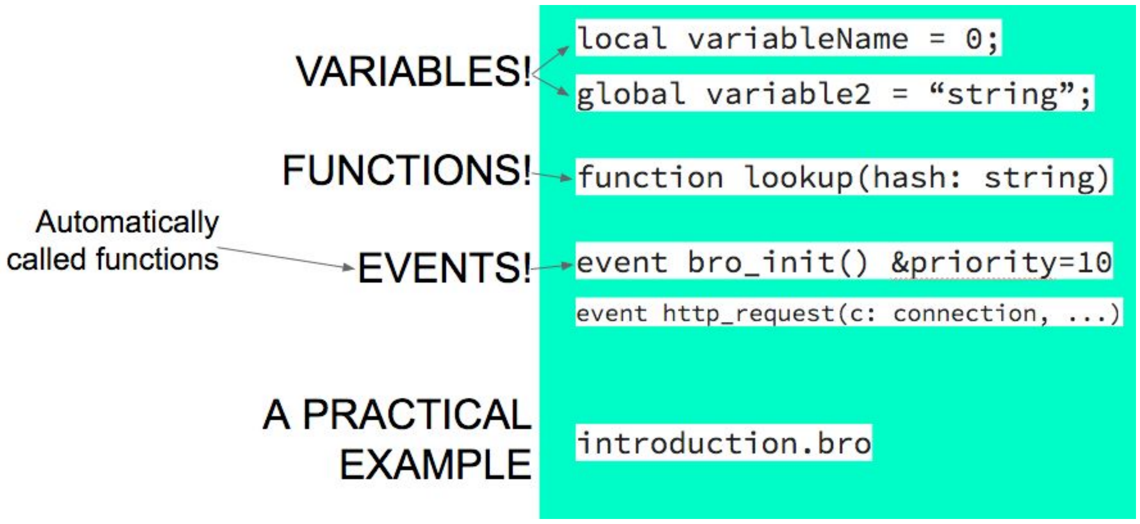
For easy-manage the large amount of log files and informations inside them, Bro provide an interesting command: bro-cut, that helps human to read Bro logs; indeed by simply writing:

```
cat dns.log  | bro-cut id.orig_h id.resp_h query
```

All the rows of the file *dns.log* are displayed but keeping only the columns specified, that in this case are *id.orig_id*, *id.resp_h* and *query*. That represents the *origin address*, the *response address* and the *dns query*. (Each row represent a record, each column an attribute)
Many other options provided to filter the logs content are:

- **`head/tail -n`** `outputs the first/last n lines,`
- **`uniq`** `filter repeated lines,`
- **`sort`** `very useful for sorting/grouping and filtering`

The combination of these useful commands, as we can see in the example, will help the user to restrict its observation, making it more specific.
Instead, if the user prefers to customize the behavior; to target some attacks, some kind of network activity or just to add useful logs, the user can implement its own *Bro Script* using the *Bro scripting language*.
A simple overview:



```
local variableName = 0;
global variable2 = “string”;
function lookup(hash: string)
event bro_init() &priority=10
event http_request(c: connection, ...)
introduction.bro
```

Lab Overview

We organized the Lab bringing at first an introduction to Bro itself, very similar to the introduction of this report, and then providing a useful view of the current tool used to manipulate the logs generated by Bro. We also provided a view of a Sublime Text’s plugin that enable syntax highlighting for the Bro scripting language (useful to present and to program the bro language - essentially to avoid notepad programming).
The most important part of the lab was the bro language (because bro is fully customizable through bro scripts) that we introduced too with a simple *“introduction.bro”* script.
The introduction script contains various events and variables of simple and complex types; basically we were showing that it is easy to do some monitoring on the net, examples:

- counting how many people ask for *“wikipedia.org”* domain name and log their IPs
- handling *bro_init* and *bro_end* events and purposes
- monitoring for DOS attacks that aims at filling the queue of a service on a certain host with a great number of established TCP connections
- count how many HTTP connections happen and log them
- show the IP and URI of any host that is downloading a file or viewing a pdf through HTTP

The introduction covered also the declaration of variables, functions, events, properties and time-flushing table entries (EG: &create_expire=5).

After the introduction we showed how to setup two different network attacks and the relative detections, these are:

- SQL Injection
- DNS Injection

Equipment

We decided to present these 2 demonstrations by using 3 VMs of a Debian Distribution, equipped with all the programs needed and configured for the Lab session.
These VMs were:

- Attacker (used in SQL Injection, DNS Injection)
- Server (used in SQL Injection, DNS Injection)
- Victim (used in DNS Injection)

As you can see, the Victim VM was used only for the second attack demonstration.

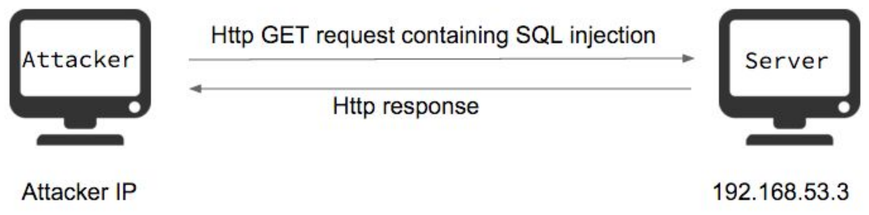
The software used in this lab is mainly common software, part of GNU/Linux.
Except some packages, like: the *dsniff* package that was installed on the attacker machine to attack in the second scenario. Others are: sublime text 2 and bro.
Furthermore, in every VM we installed Sublime Text with the bro scripting plugin (for syntax highlighting), then we configured a static IP for every VM and installed Bro on Server and Victim VMs (for detecting the attacks).
Then we set up the DNS server on the server VM (for the 2nd attack) and we configured the victim to use the the newly created DNS server (modifying the */etc/resolv.conf* configuration file).

And this is all we used to make the lab.

SQL Injection

We deployed a server VM with enabled apache web server and an *“injection.php”* file in the root directory of the web server. The php script is vulnerable to SQL Injection through GET requests because of missing sanitization of the input. That will go directly into the SQL query, forming the statement imparted to the SQL server.
The server VM has a static address: 192.168.53.3
The attacker VM is the machine supposed to attack the server’s PHP page, on that we showed how to do injections through the web browser which already suggest some values on the web page forms (to do the sql injection).

Network setup overview



We did reasoned on the fact that the injection strings usually have common patterns (like the presence of ‘, and, or, =), so we started from those similarities to write a simple Bro script for detect this kind of attack.

We then assigned the writing of the detection script as an exercise for the students, for helping them we provided a template with the empty bodies of the functions that had to be filled to complete the exercise.

An example of our solution

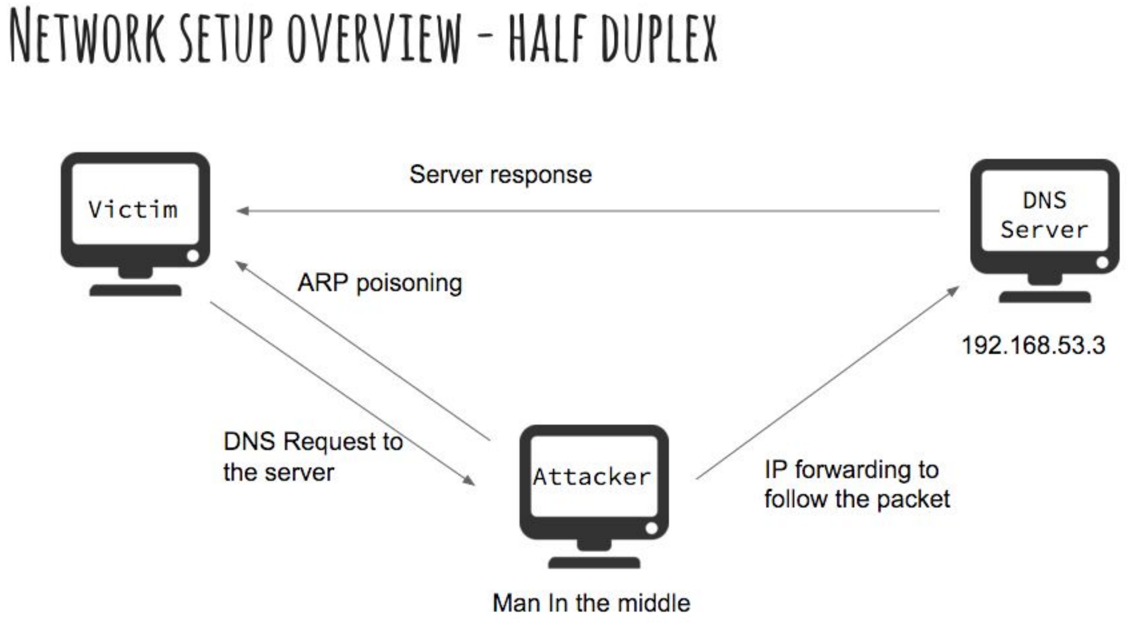
```
#array of patterns the sqli
global patterns = set("+or+","+and+","'");

#event
event http_request (c: connection, method: string, original_URI: string,
unescaped_URI: string, version: string){

     for( p in patterns){
          if( p in to_lower(unescaped_URI)){
               print fmt("SQLInjection detected from %s",c$id$orig_h);
               break;
          }
     }
     print fmt("%s -> %s",c$id$orig_h,unescaped_URI);
     print "";
}
```

After a while we provided also the solutions and we showed them that effectively the script does work and detect SQL Injections. We also noted that the solution isn’t perfect because it can generate many false positives. So we challenged the students to improve our solution.

The simple basic solution provided checked that the unescaped URI was not containing the symbols: ‘, ‘and’ or ‘or’.

DNS Injection

The second attack (and detection) of the day was the DNS Injection. The network setup for this attack was a bit more complex than the previous one: in this situation we have a client (Victim VM) that issue a DNS request to a server (Server VM), this request is spoofed through MITM from the attacker (Attacker VM).


NETWORK SETUP OVERVIEW - HALF DUPLEX
Victim
Server response
DNS Server
ARP poisoning
192.168.53.3
DNS Request to the server
Attacker
IP forwarding to follow the packet
Man In the middle

In order to make this attack work on a single physical machine, a little tweaking was necessary and a 1 second network delay has been added to the packet output by the server VM, to prevent it from answering before the attacker could.
We also explained to the students the behavior of the attacker script (provided by our team).

the attacker poisons the ARP table of the victim (using arpspoof) so that the victim thinks that the server has the attacker’s mac address. In this way the attacker can sniff the traffic of the victim. Then, to generate fake DNS replies, dnsspoof from the dsniff package was used.

In the example scenario we asked the students to generate the DNS requests from the victim machine with the *nslookup* command (that is a raw DNS client). It was useful to show how the reported IP address changed when the spoofing script was running on the attacker machine.

Then we showed how to detect the spoofing exploiting the key fact that in a spoofing attack as this, it is possible to detect a double answer from the server. The detection was implemented in an example bro script that we developed.

An example of our solution

```
#the table of address
global dns_cache: table[string] of addr &create_expire=5 sec;
#the event
event  dns_A_reply(c: connection, msg: dns_msg, ans: dns_answer, a: addr) {
     local s:string;
     s=(fmt("%d",msg$id));
     if(!(s in dns_cache)) {
          dns_cache[s]=a;
     }
     else{
          if(dns_cache[s]!=a){
               print "DNS Injection detected";
               #print fmt("%s,%s",dns_cache[fmt("%s",c$id)],a);
          }
     }
     print fmt("A record:%s %s",a,msg);
     print "";
}
```

We also showed how to improve the attack to fool the detection. Basically the improvement consists in arping the server to make him think that the victim has the mac address of the attacker, in this way the attacker intercepts both messages (full-duplex hijacking), the ones from the victim and the ones from the server. Then, disabling the *ipv4* forwarding feature in the Linux Kernel, the attacker creates a *“black hole”* and the messages coming from the victim never reach the server.
It is important to note that the second (enhanced) script does fool the *“iceweasel”* browser (it is a minimal and more secure version of firefox), while the basic exploit is not able to fool iceweasel (because of his embedded detection mechanisms), the second empowered script does work. Also note that with the second script, the detection written in *bro* does not work anymore because there is only one reply from the server.

Conclusions

The laboratory covered the very last lesson (Monday PM) and the topics covered were very advanced for part of the audience that was not properly able to execute a successful SQL Injection. The scripts and exercises that we brought required a certain level of intuition and a very basic level of bro programming understanding, excluding the “Hello World” test codes.
Preparing the laboratory session was also very useful for our knowledge, as Bro is very powerful and having a little experience on the Bro scripting language can be very useful for many purposes.