# 12TH NETWORK SECURITY LAB
# *BRO NIDS*

Group 3
*Giulio Dallatorre*
*Tomas Bortoli*
*Alex Mariotto*

## We are going to present...

An introduction to Bro NIDS

- main features
- tools
- Language

Two examples attacks

- SQLI
- DNSI

Two example of detections
- SQLI Detect
- DNSI Detect

## Overview - What is Bro?

Bro is a passive open-source network traffic analyzer written in C++.
It is primarily a security monitor that inspects all traffic on a link in depth for signs of suspicious activity.

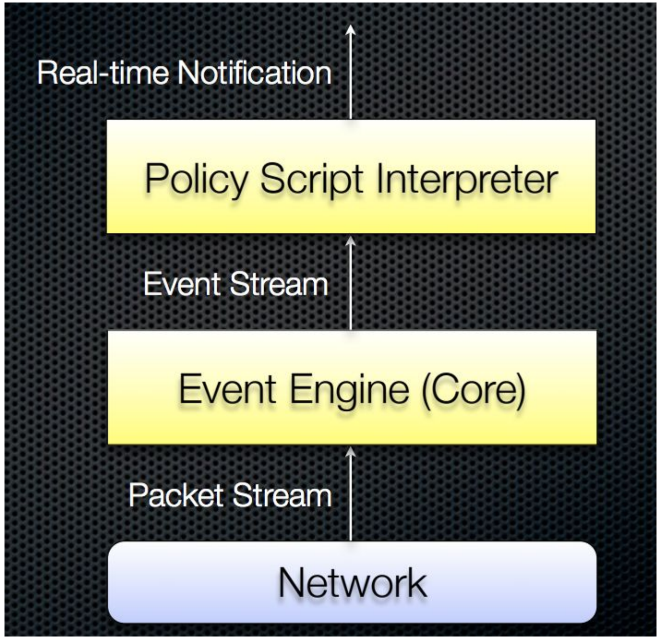
It is fully-customisable; the owner can perform specific observations of events and notify (log) interesting events.
It can also be used to evaluate the performances of the network.

## Interesting Features

By default bro is passive, but it can also be configured to be active (by writing scripts).

It does support signature based detection but also anomaly based detection.

Bro itself is a framework to work with, to design defensive solutions. Basically it provides a high level interface to most of the traffic generated by your network through its scripting language, **bro** (slide 6).



## Bro logs

By default bro generates a lot of high-level logs about data going through the network.

Some examples >

**conn.log** (log TCP/UDP/ICMP connections)

**dhcp.log** (log DHCP leases)

**dns.log** (log requests and responses)

**ftp.log** (ftp activity)

**irc.log** (log IRC commands and responses)

**files.log** (log file transmission under different protocols)

**reporter.log** (Internal error/warning/info messages)

**packet_filter.log (**List packet filters that were applied)

**ssl.log**

## Tool To manage logs

**bro-cut** helps human reading of bro logs,

EG: **cat dns.log | bro-cut id.orig_h id.resp_h query**

gives as output all the rows, by keeping only the columns named as id.orig_h, id.resp_h, query. So, in simple words: origin_address, response_address, dns_query

**head/tail -n** outputs the first/last n lines,

**uniq** filter repeated lines,

**sort** very useful for sorting/grouping and filtering (a lot of options)

these tools can be combined for retrieve more specific log overview.

## Bro Scripting

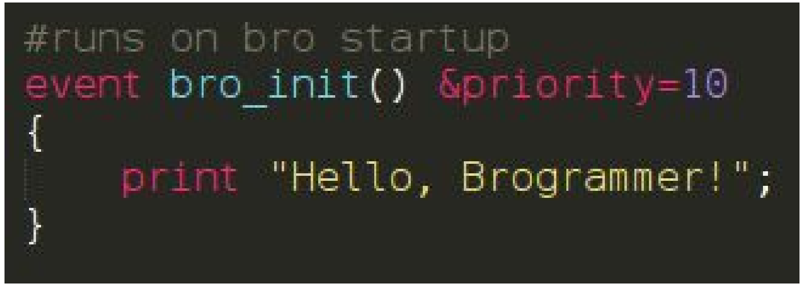
Bro provides a scripting language designed and developed by the bro developers. It's aim is the processing of the network traffic; bro scripts are given in input to bro when started and are then interpreted by the internal interpreter.

- easy and fast to develop
- syntax highlighting supported on Sublime Text (by plug-in)
- need support from online documentation to discover needed event and parameters for a certain packet

## Technical Features

Bro scripting is very similar to c++.

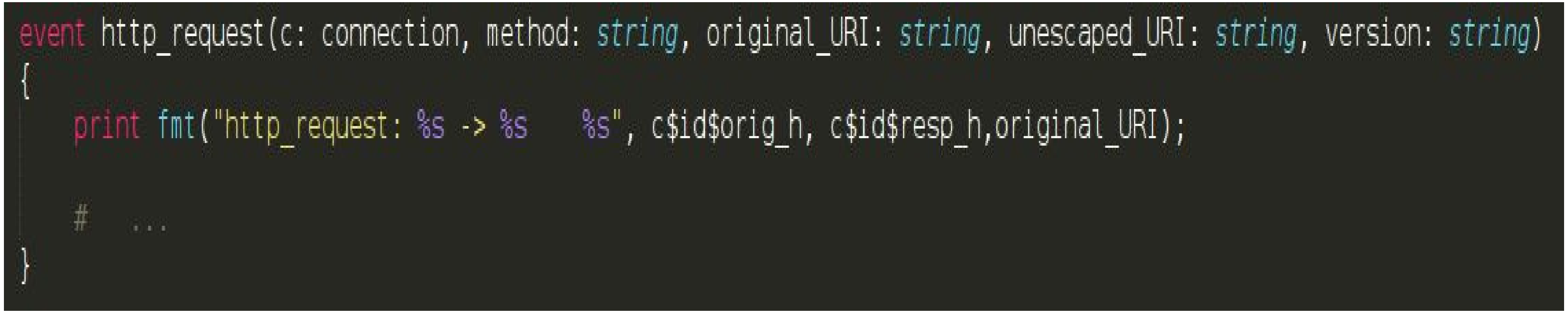
- event driven, to catch the packets of interest
- supports many default types
- allows special properties to be set on variables



```
#runs on bro startup
event bro_init() &priority=10
{
    print "Hello, Brogrammer!";
}
```

It’s easy to run by simply typing

**`bro -i $interface myscript.bro`**



```
event http_request(c: connection, method: string, original_URI: string, unescaped_URI: string, version: string)
{
    print fmt("http_request: %s -> %s    %s", c$id$orig_h, c$id$resp_h,original_URI);

    #  ...
}
```

**This little example show a bro script that logs the sender & receiver IPs and the requested URI of all the http requests in transit on the net**

## VARIABLES!

```
Local variableName = 0;
Global variable2 = “string”;
```

## FUNCTIONS!

```
function lookup(hash: string)
```

Automatically called functions

## EVENTS!

```
event bro_init() &priority=10
event http_request(c: connection, ...)
```

## A PRACTICAL EXAMPLE

```
introduction.bro
```

Questions?

Ready?

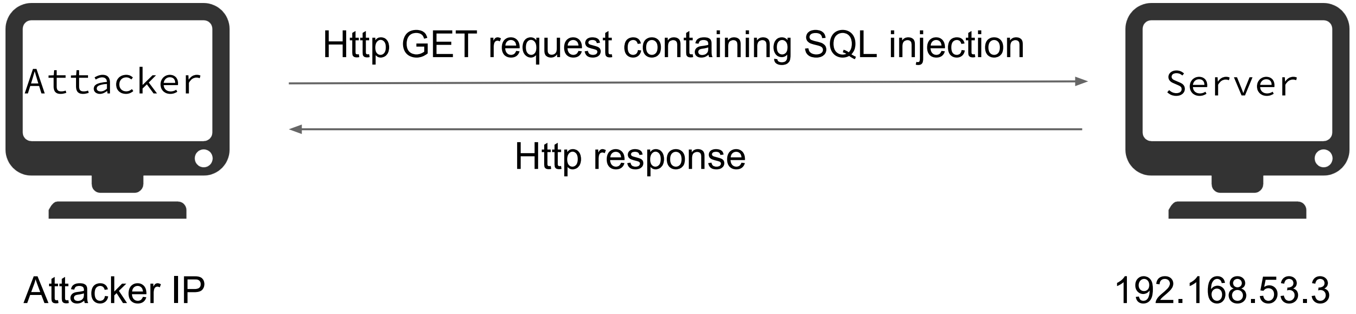
Go with the SQL Injection!

## SQL Injection Overview

SQL Injection is a technique of injection that manipulates the input query issued to the sql server, fooling the PHP code.

Usually for this kind of injections, some defined patterns are used: especially some characters…...

## Network setup overview



## We need to setup the attack

For this purpose we need to run the following VMs

- *Server* (net address 192.168.53.3)
- *Attacker*

By connecting to the server with http we can see a web interface in which it will be simple to do some SQL Injections. (192.168.53.3 /injection.php)

The interesting part is detecting them using Bro language so we’ll write a simple detection script.

# Do you remember? *Bro* is event-oriented

Detect an http request event and some of its parameters:

1. connection info (see next slide)
2. method (EG: “GET”, “POST”, ..)
3. original URI
4. unescaped URI
5. version

These will give us the informations that we need to detect potentially malicious HTTP GET requests.

For the other infos, check bro resources

```
event http_request (

    c: connection,

    method: string,

    original_URI: string,

    unescaped_URI: string,

    version: string

) { ….?sniff function body spoof¿…}
```

https://www.bro.org/sphinx/script-reference/proto-analyzers.html

# The connection info parameter

Out connection parameter, comes with many interesting features, for example:

- ***c$id$orig_h***
- ***c$id$orig_p***

And obviously:

- ***c$id$resp_h***
- ***c$id$resp_p***

Connection  object notation [like '.' in Java or '::' in C++]

We have a variable that lists the IP addresses:

- Sender IP address
- Sender port
- Receiver IP
- Receiver port

More on: https://www.bro.org/sphinx/scripts/base/bif/bro.bif.bro.html

## Other interesting methods & constructs

***to_lower()*** -> transforms the Uppercase words to lowercase ones, where ***to_upper()*** is the inverse.

***fmt("Hello %s!", "world")*** -> format strings, extremely similar to the printf.

Regarding the print, for, if and variables the syntax is the same as C++.

Fill the methods in the

mySqlDetect.bro script

In the **SERVER** VM.....plz

## Workflow Overview

### Server

*1* - Start bro with the detect.bro script. Type:

**“ sudo su “**

**“ bro -Ci eth0 mySqlDetect.bro “**

*4* - Then check the BRO console output.

### Attacker

*2* - Open the browser and load the insecure webpage by connecting to the webserver (using its IP).

**“(*server_address*)/injection.php”**

*3* - Execute some queries on the example webpage, try to execute some SQL injection.

## Observations

Have you noticed the false positives given by the application?

With the command “ **subl sqli/detect.bro** “ you can edit the script (our). Sublime text will have syntax highlight for BRO scripts.

How would you make the script better?

## 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 "";
}
```

## Ready?
## Go with the Poisoning!

## DNS Poisoning Overview

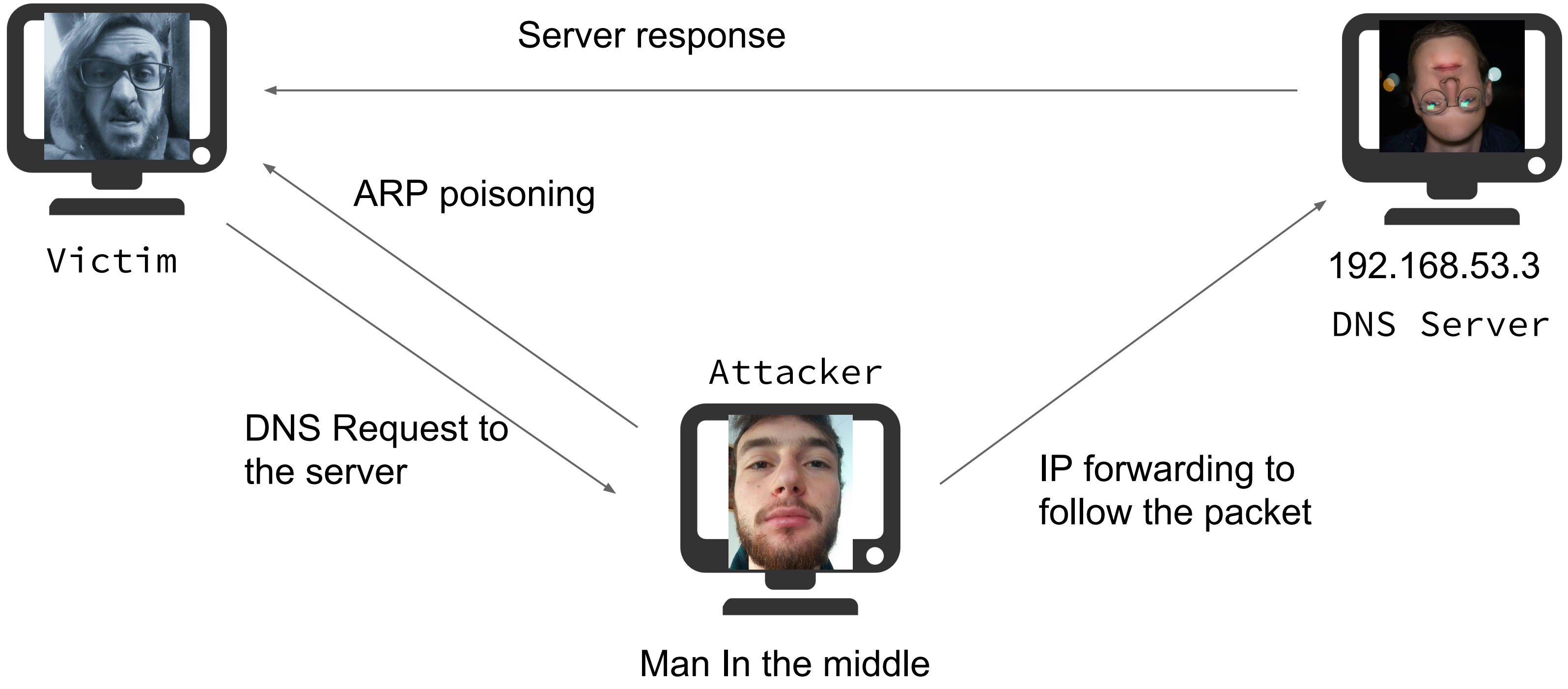
DNS Poisoning is another interesting technique that acts as MITM attack.

The attacker makes the victim think that the required server is on a different IP address.

This method will “poison” periodically the DNS cache table of the victim using DNS requests and so permits to associate a different IP address to a certain nameserver.

And of course this different IP should be the attacker’s one.

## Network setup overview - half duplex



## We need to setup the attack

For this attack we need again the ***Server*** and ***Attacker*** VMs plus the ***Victim*** one.

Using the *Victim* open a terminal window and then execute the command “**nslookup brolab.com**”. Make sure that you get an answer pointing to the *Server* VM.

## Other interesting features! DNS_MSG

Let’s play with data structures:

```
dns_cache: table[string] of addr &create_expire=5 sec;
```

The previous stmt declares a table that contains addresses and uses string as indexes. Furthermore the expiration time of any entry is set to 5sec. It’s a hint…

```
event  dns_A_reply(c: connection, msg: dns_msg, ans: dns_answer, a: addr){
```

This will be called in case of a DNS A type reply.

Fill the methods in the mySpoofDetect.bro script

In the VICTIM vm.....plz

## VICTIM

*1* - Open a new terminal window, execute the bro script dns_spoof_detect.bro

**“Sudo su “**

**“bro -i eth0 mySpoofDetect.bro ”**

## ATTACKER

*2* - On a terminal grant yourself “super user” privileges by running “**sudo su**”.

*3* - Then execute the script “**./inject.sh**”

*4* - execute again the “**nslookup brolab.com**” command.

## 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 "";
}
```

## Observations

Has the address been spoofed?

What does the *Victim*’s BRO console says?

Terminating the attack script will the cache return working normally? Why?

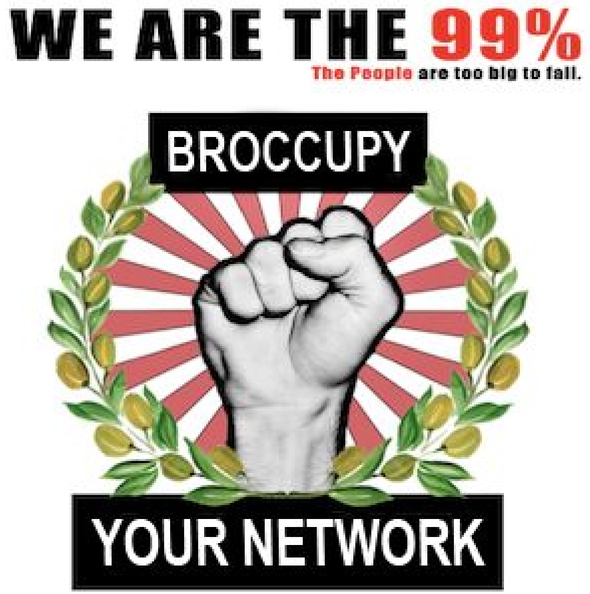
Is Iceweasel browser is affected by the attack? Why?

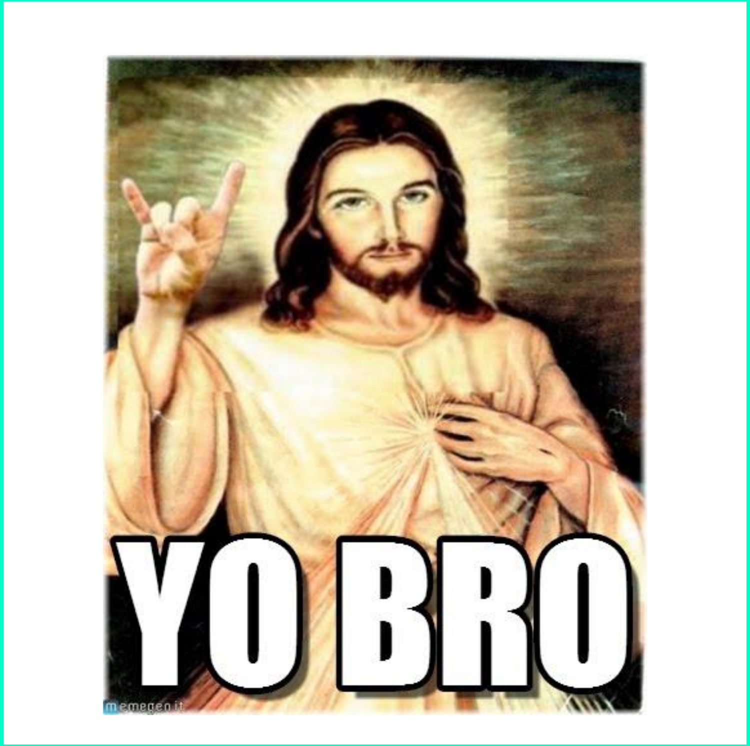
## So...

We have seen that Bro provides a rich framework to monitor and detect unusual behaviours inside a network.

If you feel good with programming the only limit is your fantasy.

Find more on: https://www.bro.org




YO BRO
memegen.it