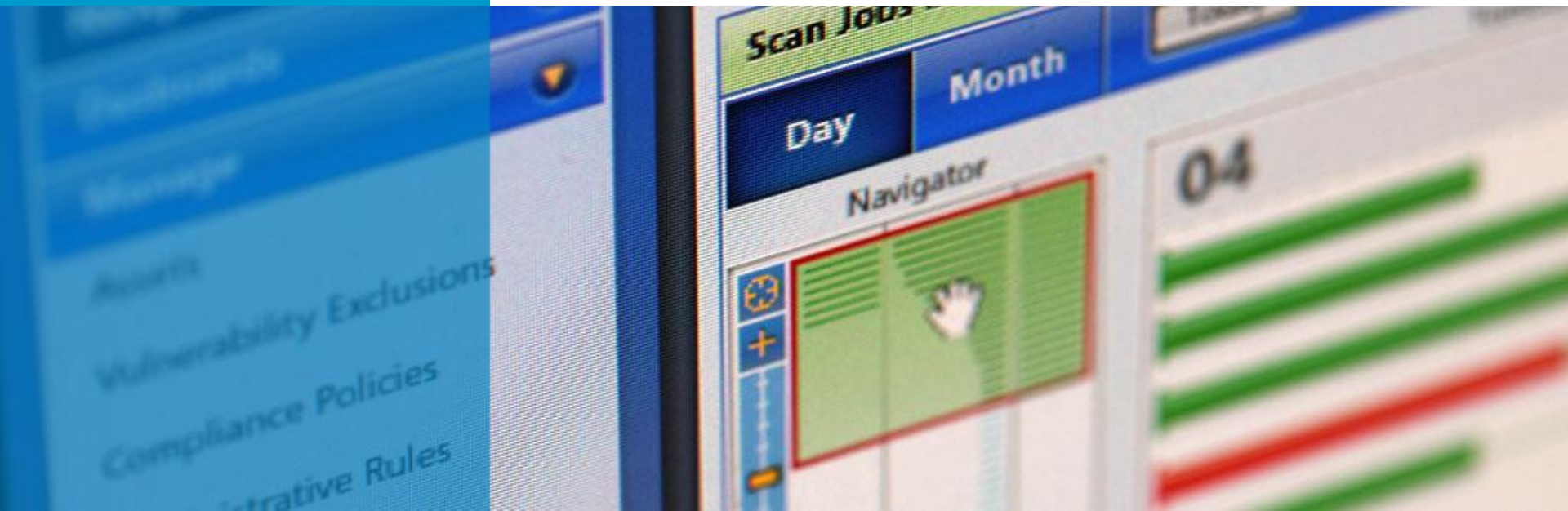
RAPID7

Web Application Security Payloads

Andrés Riancho

SecTor 2010, Toronto, Canada.



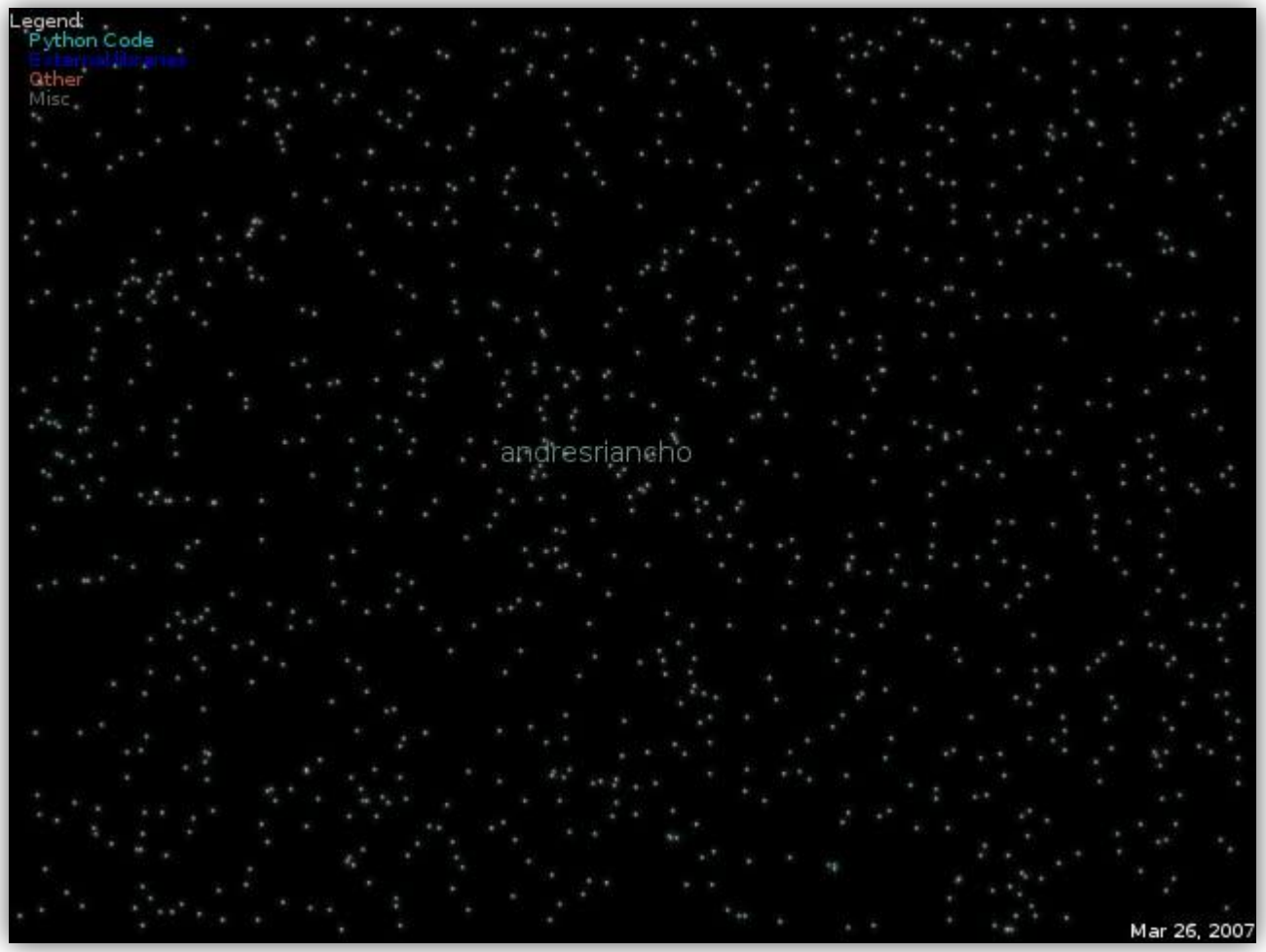
andres@rapid7.com$ whoami

- **Director of Web Security** @ Rapid7
- **Founder** @ Bonsai Information Security
- Developer (python!)
- Open Source Evangelist
- Deep knowledge in networking , design and IPS evasion.
- Project leader for **w3af**

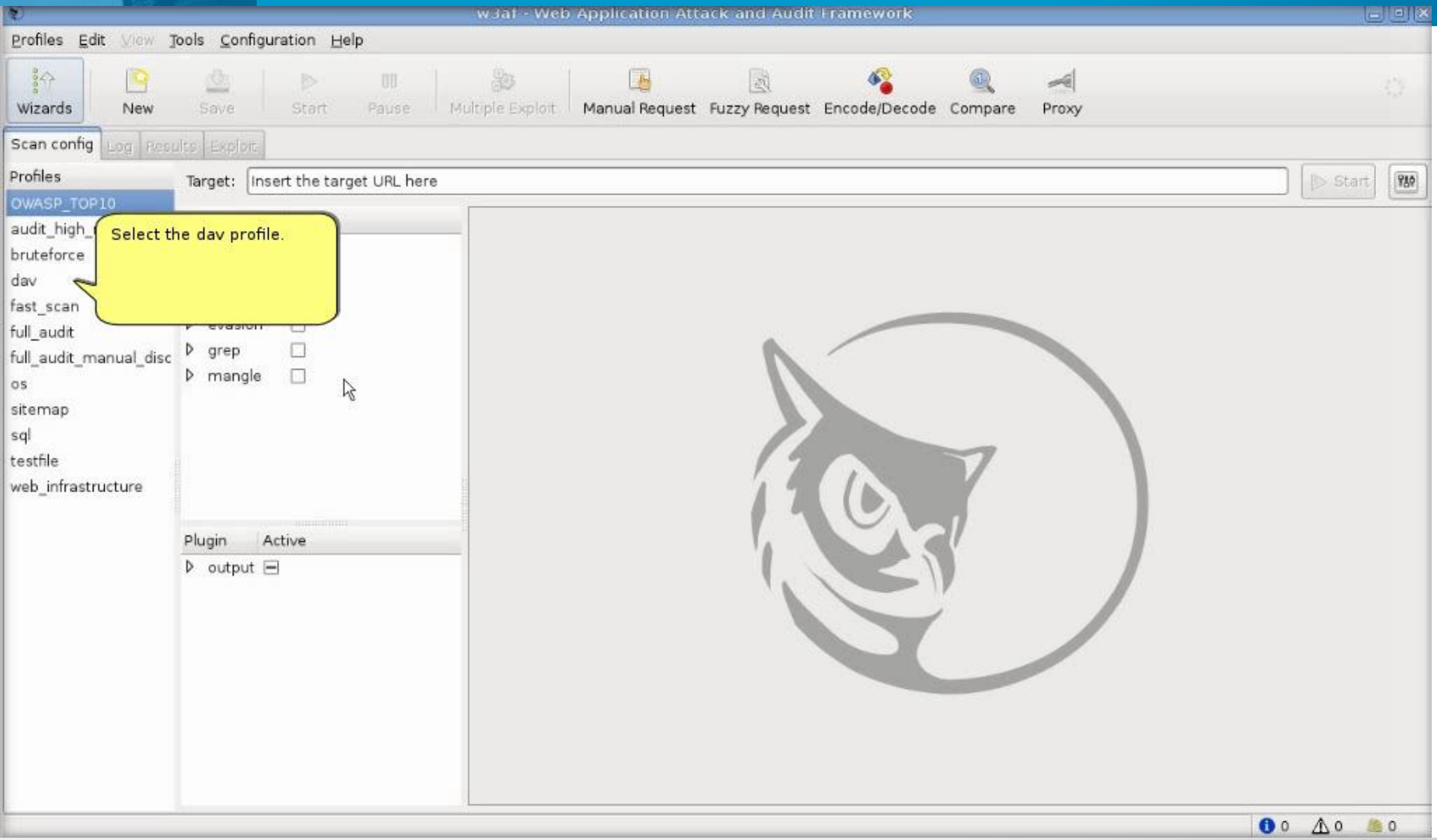
w3af

- w3af is a **W**eb **A**pplication **A**ttack and **A**udit **F**ramework
- Open Source tool (GPLv2.0) to **identify and exploit Web vulnerabilities.**
- Plugin based architecture, **easely extensible**.
- Development started in late 2006 on my spare time, at this moment we have multiple contributors from around the globe and **a full time developer @ our Buenos Aires office**.

Code Swarm



Short GUI demo



What we've achieved

- In these **four years of life**, the w3af project has achieved these goals:
  - Widely known, distributed in most (all?) hacking live-cds
  - Packages for most linux distributions
  - A relatively low false positive rate (when possible)
  - Good link and code coverage
  - A low false negative rate.

- **We still have much to acomplish!**

The incident that triggered our research

- The guys at Bonsai were working on a **Web application penetration test** and they **identified an arbitrary file read** in a PHP application.
- After **two hours** of reading different files and trying to find something that would help us elevate privileges. Nothing interesting was found.
- **One more hour**, and we were able to find an application directory that wasn't linked, where we identified a way to upload files that allowed us to get **command execution with an un-privileged user (www-data).**
- Even after that, **we had to work for some time** to get all the information out from the database and get root (mysql password == root password).

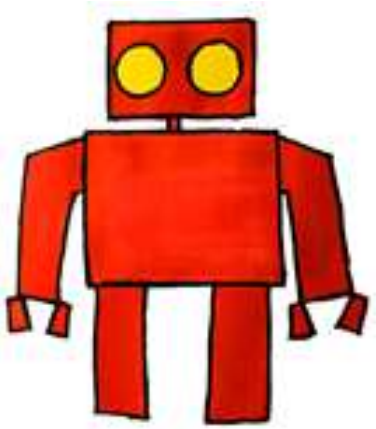
The incident that triggered our research

- During this experience we noticed that:
  - Exploitation frameworks like Core Impact or Canvas provide **"exploits and payloads" to use in best case scenarios**, in other words, when there is control on the execution flow ("exploits for buffer overflow").
  - **None of the currently available tools**, Open Source or Commercial, have any post exploitation techniques we could apply to **Web application vulnerabilities in order to escalate privileges.**

The reasons

- Exploitation frameworks are focused on memory corruption exploits because they **were the most important vulnerability class**.

- **Attention has now shifted to Web applications**, which are different because they only allows us, depending on the vulnerability, **to interact with the system in a particular way**:
  - Read a file
  - Write a file
  - Control a section of a SQL query
  - Execute user controlled source code
  - Execute operating system commands



Our solution

- **We approached this problem** keeping in mind the **capabilities that each Web application vulnerability** exports. These are some of the questions that we asked ourselves:

- What's possible if we're only able to read files?
  - I want the Apache config files!
  - And the .htpasswd files also!
  - I would like to see the remote process list, is that possible?
  - What about open TCP and UDP connections?

- What if we're able to upload images to the webroot?
  - If we're also lucky enough to also have a local file include vulnerability, how can we combine both?

Web Application Security Payloads

Design

- Each exploit exports "**system calls**", which are then used by the payloads:

Exploit	Exported system calls	Emulated system calls
Local file read	read()	
Local file include	read()	
OS Commanding	execute()	read() , write() , unlink()
DAV Shell	write()	execute() , read(), unlink()
File Upload	write()	execute() , read(), unlink()

- Each syscall acts as an **abstraction layer**, allowing the payload to run without knowing/caring which exploit is in use.

Design

- Payloads are usually **short code snippets** that use a couple of system calls and have specific **knowledge about which files to read** and how to extract information from them:

```
pci_list.append('1233')
pci_list.append('1af4:1100')
pci_list.append('80ee:beef')
pci_list.append('80ee:cafe')

for candidate in candidates:
    file = self.shell.read('/sys/bus/pci/devices/'+candidate)
    pci_id = parse_pci_id(file)
    pci_subsys_id = parse_subsys_id(file)
    for pci_item in pci_list:
        if pci_item in pci_id or pci_item in pci_subsys_id:
            result['running_vm'] = True
```

Demo #1: “users”

Baby steps

Sinergy between payloads

read() System call to read files

users Payload that reads "/etc/passwd" and identifies home directories

interesting_files This payload uses the home directories and a list of interesting filenames to search for passwords.

Demo #2: “interesting_files”

Sinergy between payloads

Design

- Payloads can **take decisions based on facts that were saved to the knowledge base during the scan**:
  - Identified vulnerabilities
  - Remote Web server vendor
  - Remote operating system
  - Found URLs

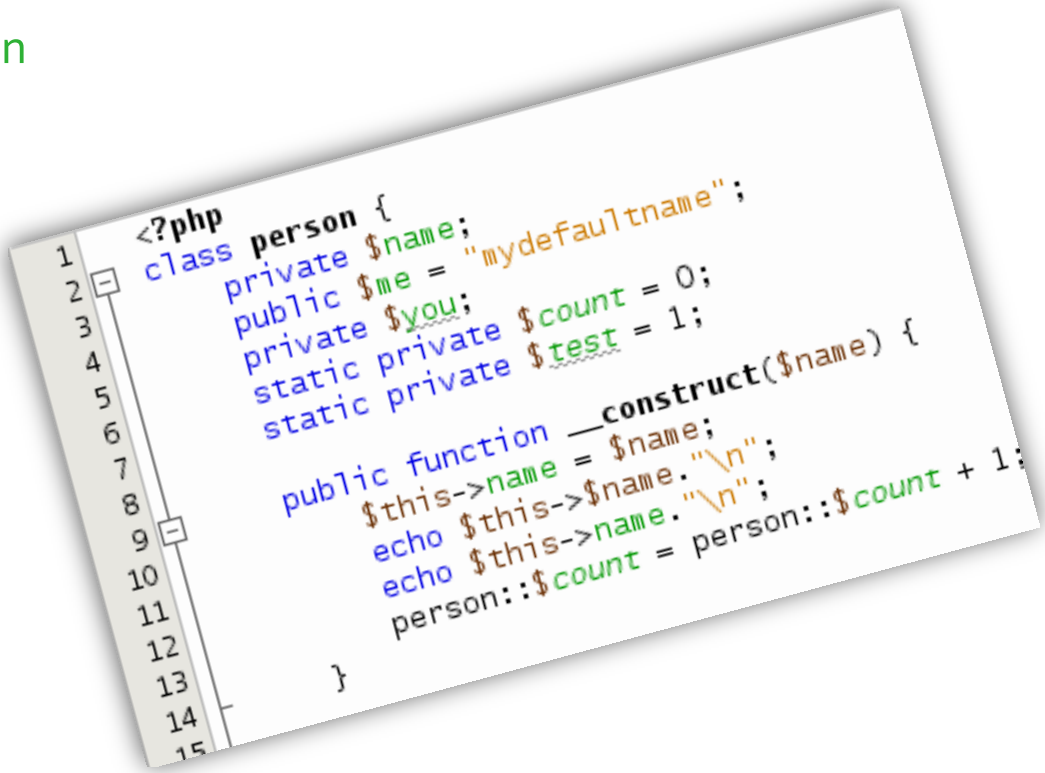
- This is one of the biggest advantages of having everything integrated into w3af!

The "get_source_code" payload

```
apache_root_directory = self.exec_payload('apache_root_directory')
webroot_list = apache_root_directory['apache_root_directory']

url_list = kb.kb.getData('urls', 'urlList')

for webroot in webroot_list:
    for url in url_list:

        path_and_file = getPath( url )
        relative_path_file = path_and_file[1:]
        remote_full_path = os.path.join(webroot,relative_path_file)

        file_content = self.shell.read(remote_full_path)
        if file_content:
            self._save_file_locally(remote_full_path, file_content)
```

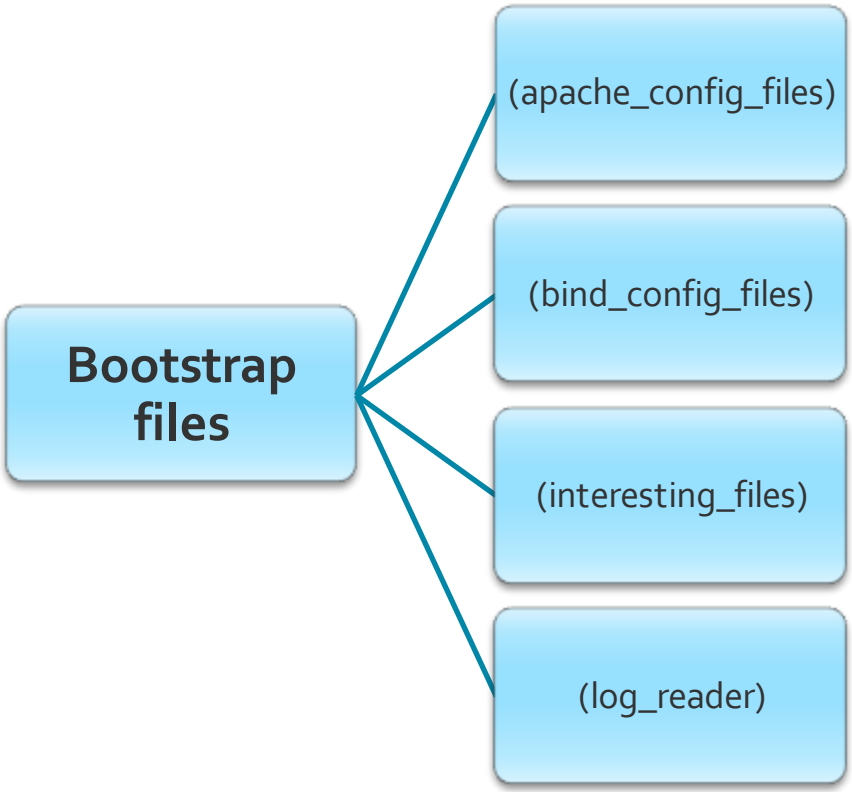
Demo #3: “get_source_code”

w3af integration



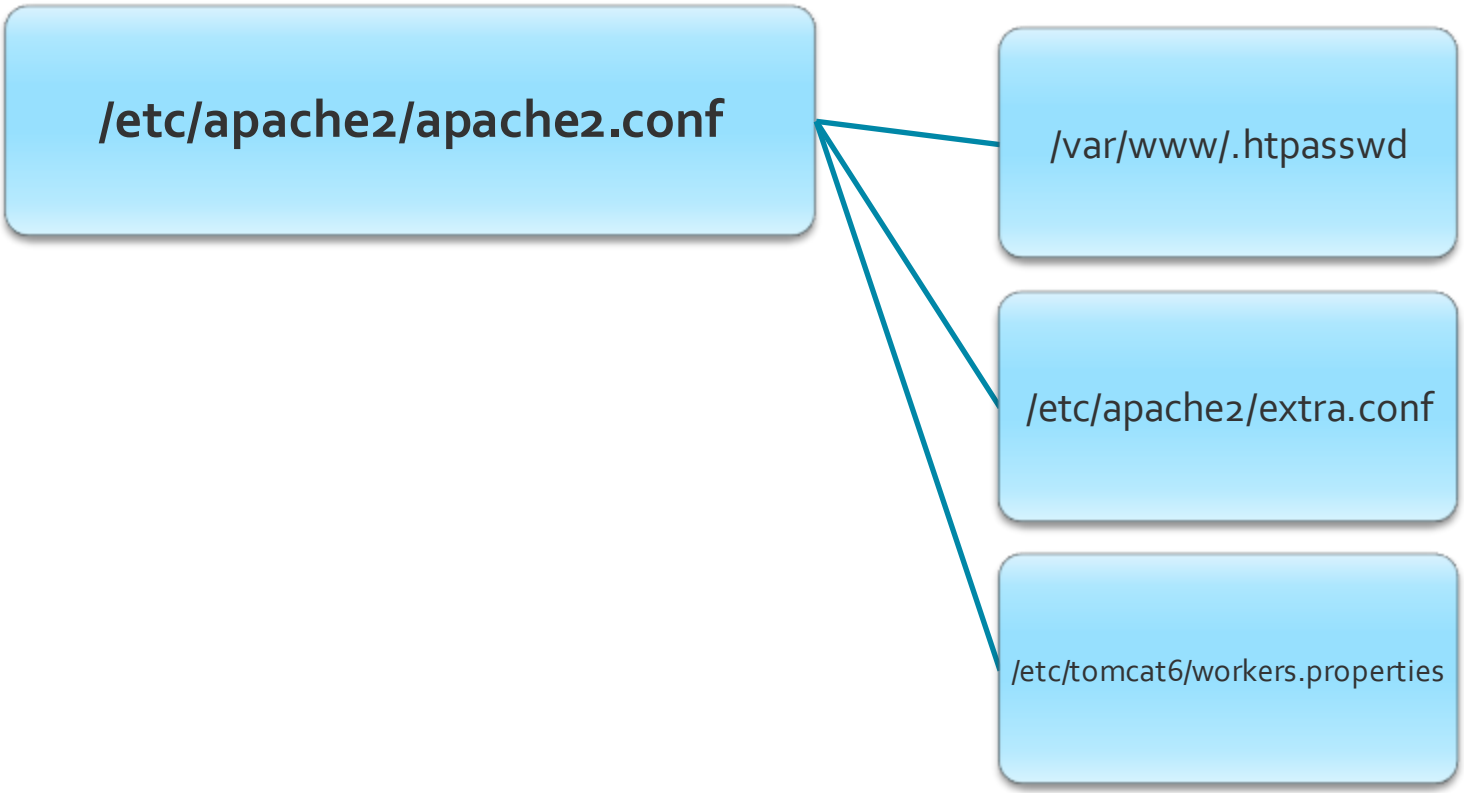
A remote file system spider

- The last example is one of the **simplest but most effective payloads we've developed: "spider".**



A remote file system spider

- The last example is one of the **simplest but most effective payloads we've developed: "spider".**




directory
stealer
apache
ssh
mysql
current
get
vm
shell
honeypot
allowed
interesting
running
read
ldap
kernel
user
log
gcc
mail
hosts
processes
arp
ssl
uptime
modsecurity
run
info
version
netcat
hostname
fingerprint
dhcp
root
cpu
installed
ftp
htaccess
cache
filesystem
users
tcp
list
hunter
hashes
domainname
modules
firefox
reader
login
interactive
rootkit
kerberos
iis
route
group
smb
udp
svn
spider
dns

And when we **can execute OS commands…**

- Great! We found a way to execute operating system commands using our web application payloads that run with low privileges, **now what?**

- When we're able to execute OS commands **everything is simpler**. In these cases, w3af provides the following payloads:
  - metasploit
  - msf_linux_x86_meterpreter_reverse
  - msf_windows_meterpreter_reverse_tcp
  - msf_windows_vncinject_reverse
  - w3af_agent

Demo #4: metasploit integration

msf_linux_x86_meterpreter_reverse

w3af agent

- The w3af agent allows us to **route traffic through the compromised host** without any effort.

1. w3af **uploads an agent client** to the remote host
2. The agent **client connects back**, and the TCP connections are kept alive to route traffic.
3. **w3af starts a SOCKS daemon** in the local machine, which is the entry point for all connections that the user wants to forward.

Demo #5: “w3af_agent”

Routing traffic through the compromised host

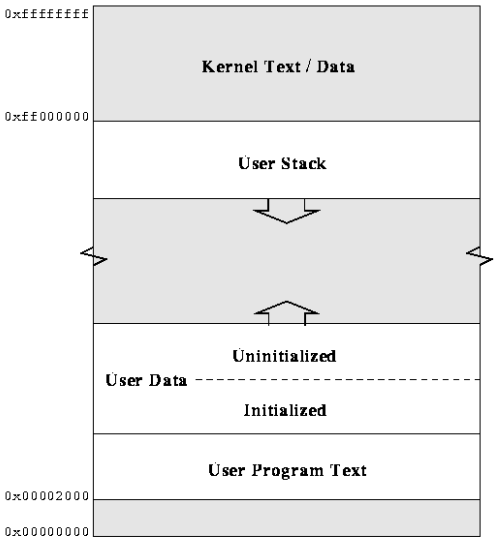
Syscall hooking

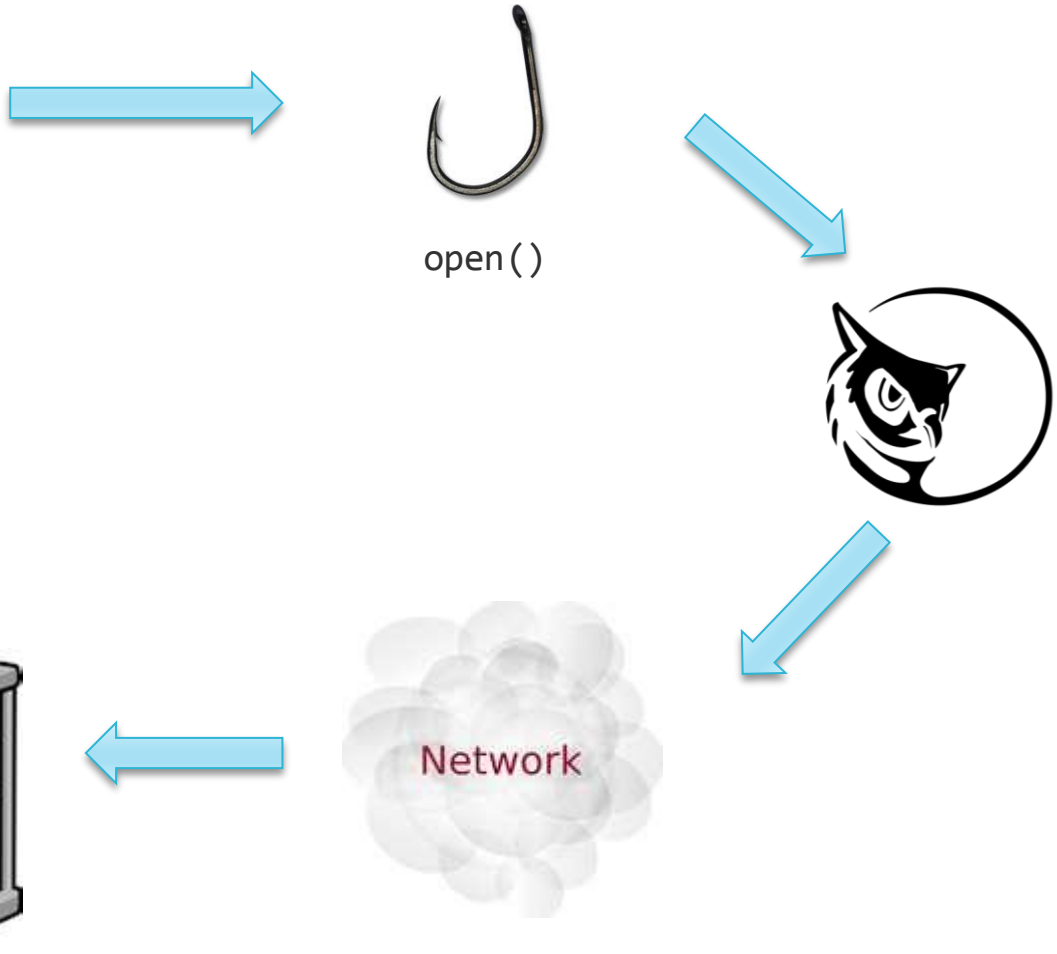
- Syscall hooking using **ptrace()** is a **research in progress**, for which we only have a **small PoC**, but I wanted to explain it here to **get feedback and new ideas.**

- **The initial idea** we had with Lucas Apa (the main Web application security payload developer) was to **create a framework that would hook into a process' and forward it over the network** to the remote server using the Web application exploit.

- Using this method, **we would be able to run any software installed on the host running w3af in the remote box**. A simple example would be "clamav".

Syscall hooking





Syscall hooking

SUBTERFUGUE

- In a very distant past, I played with **subterfugue**: "a **framework for observing and playing with the reality of software**; it's a foundation for building tools to do tracing, sandboxing, and many other things. You could think of it as **"strace meets expect"**."

- Which is a great software for hooking into a process using ptrace and modifying it's state, but has **two big issues:**
  - Not supported by the orginal developer anymore
  - Doesn't work in 64bit arch.

Syscall hooking

```
# Called before linux's read() syscall
def callbefore(self, pid, call, args):
    m = Memory.getMemory(pid)
    arg_mem_addr_path = args[0]

    filename = m.get_string( arg_mem_addr_path )
    # Calling the "read" syscall of one of w3af's exploits
    local_filename = self.shell.download( filename )

    area, area_size = m.areas()[0]
    m.poke(area, local_filename + '\0')

    # Rewrite the syscall in order to read the local file
    return (None, None, None, (area, args[1], args[2]) )
```

Conclusions and pending work

- Our objective is to make this the **standard for automatized post-explotation** of Web application vulnerabilities.
- Develop more payloads for **Windows** environments.
- Combine more payloads **and under certain circumstances**:
  - Launch a new scan against a particular resource
  - Assert new vulnerabilities
  - Exploit vulnerabilities using the increased knowledge obtained by w3af's payloads
- **Syscall priority** : when more than one syscall exists, which one should we use to communicate with the remote system? The fastest one? The one with more privileges?
- Finish first implementation of **syscall hooking** supporting the read() syscall**,** using **pinktrace instead of subterfugue**?

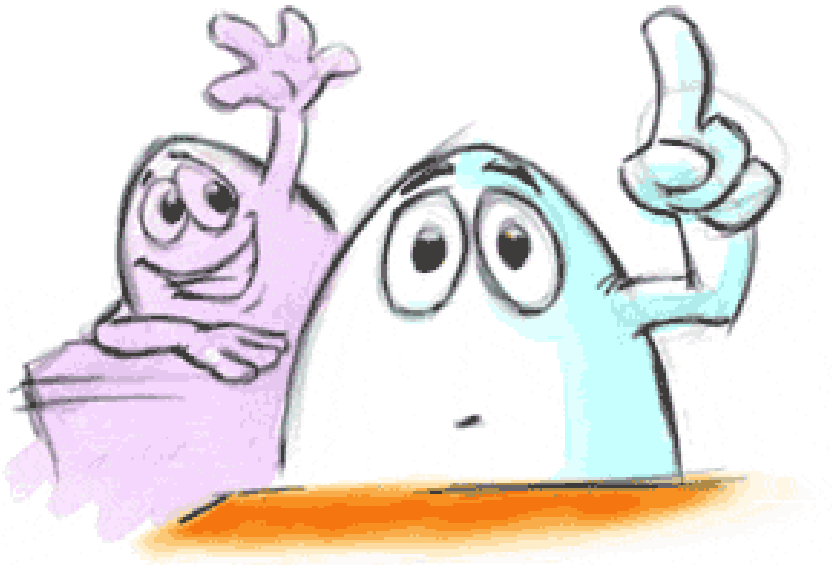
I want to contribute!

- **Got an idea for a payload?** Contact me after the talk and we'll add it to our TODO list!

- **Want to code?** The **source code** for the web application security payloads, w3af agent and metasploit wrapper **can be found in these directories:**
  - plugins/attack/payloads/
  - core/controllers/vdaemon/
  - core/controllers/w3afAgent/
  - core/controllers/payloadTransfer/

http://w3af.svn.sourceforge.net/viewvc/w3af/trunk/

¿Doubts, questions?



Thank you!

Web Application Center of Excellence, Buenos Aires, Argentina