RAPID7

Web Application Security Payloads

Andrés Riancho
Director of Web Security
OWASP AppSec USA 2011 - Minneapolis

Topics

- Short **w3af introduction**
- Automating Web application exploitation
- **The problem** and how other tools are not handling it
- **Web Application Payloads**, our solution
  - Vulnerabilities have capabilities!
  - Abstracting system calls in payloads
  - Our own SCA
  - Metasploit integration
  - Routing TCP/IP traffic
- Conclusions

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**

RAPID7

Short w3af introduction

The features and the behind the scenes story

RAPID7

Introduction to w3af

- w3af is an open source Web Application Attack and Audit Framework
  - First version released in March 2007
  - Open Source tool (GPLv2.0) to identify and exploit Web vulnerabilities
  - Architecture supports plug-ins (easily extensible)
  - Available for free download @ www.w3af.org
- w3af project is sponsored by Rapid7
  - Since July 2010
  - Full time development resources
  - Roadmap, prioritized backlog & structured development process
  - Quality assurance
  - Back office including marketing and communications

RAPID7

What we've achieved

- In these **four years of life**, the w3af project has achieved these goals:
  - Continuous, non-stop improvements in **features and software quality**
  - Good link and code coverage
  - A low false negative rate

  - Widely known, distributed in most (all?) hacking live-cds
  - Packages for most linux distributions

Stable code base and Performance

- **We still have much to acomplish!**
  - Achieve a completely **stable code base**
  - **Increase performance for the core framework features** (sending of HTTP requests, HTTP cache, analysis of responses, threading, etc.)
- Based on a **recent poll**, we're changing our roadmap to quickly achieve what users need:
  - **Stability**
  - Identify 100% of the vulnerabilities - **Scan time doesn't matter**
  - **Low False positive rate**
  - Plugin / Extension system **documentation**

The Web Application Penetration Tester issue

And how other tools are not covering it

RAPID7

Experience on a recent Web Penetration Test

Vuln!

- Identified arbitrary **file read** in PHP application

+1 hour

- Read configurations, operating system and source code files
- Found an unlinked application directory with “dead-code”

+1 hour

- **Identified arbitrary file upload** within “dead-code”
- Uploaded file to get unprivileged command execution (www-data)

+3 hours

- Accessed all DB data
- Got root privileges (mysql password == root password)

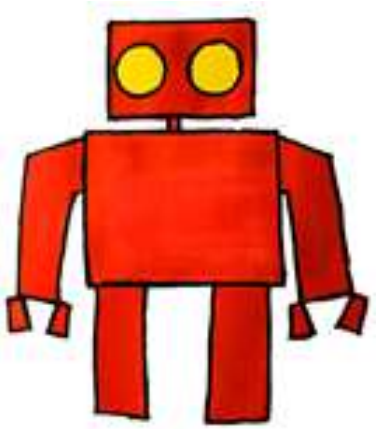
No web post-exploitation :-(

- During this experience we noticed that:
  - **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.**
  - Commercial exploitation platforms 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”).

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



Web Application Security Payloads

Helping you get root from low-privileged vulnerabilities

RAPID7

A paradigm shift in exploitation

- Which **capabilities does a Web application vulnerability export**? Two simple examples:

Web application vulnerability	Capabilities exported
Arbitrary File Read	read()
File upload	write() [often restricted to specific directory]

- **Changing our mindset** from "buffer overflow" exploits to Web exploitation with reduced capabilities, **we started to define all the actions that could be done only with read()'s:**
  - Read Apache config files,
  - Read .htpasswd files,
  - Get the remote process list,
  - Get the list of open TCP and UDP connections, and **MANY** more.

A paradigm shift in exploitation

- After identifying all actions that could be performed with read() , we **moved on to different scenarios** where we analyzed:
  - Only write()
  - Only exec()
  - write() and read() , which is usually found when there are two different vulnerabilities present.

- **Where we realized that we could emulate some syscalls using others.**

Emulating other syscalls

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

Exploit	Exported Syscalls	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.

Emulating syscalls

- Syscall **emulation** is easy in some cases, for example **read() is emulated via the execution of "cat filename"** or "type filename", depending on the OS:

```
@read_debug
def read(self, filename):
    '''
    Read a file in the remote server by running "cat" or "type" depending
    on the identified OS.
    '''
    read_command_format = self.get_read_command()
    read_command = read_command_format % (filename,)
    return self.execute( read_command )
```

- And in some other cases it is more difficult, **write() to exec()** can be **challenging** due to file system permissions, programming language configuration and the application itself.

Simple but powerful pieces of code

- 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
```

Knowledge

read()

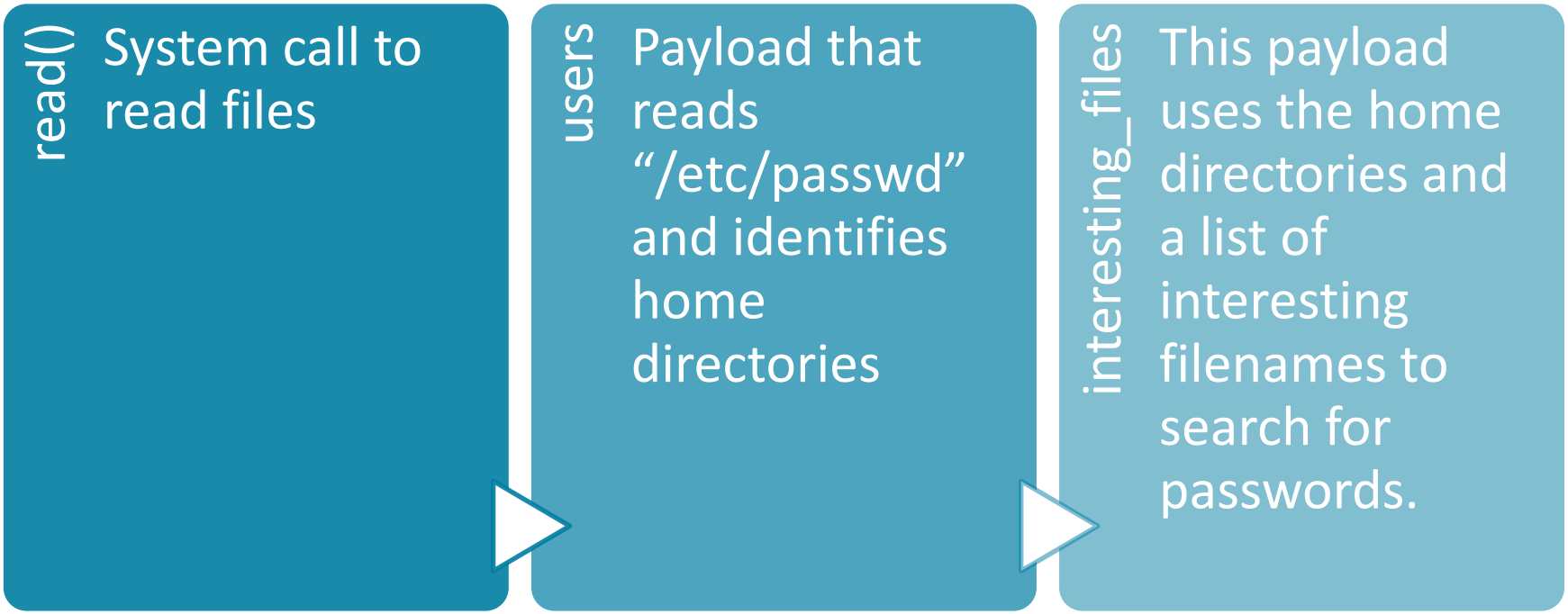
Parse

Demo “users”

Baby steps

RAPID7

Synergy between payloads



The "interesting_files" payload

```
interesting_extensions = []
interesting_extensions.append('')   # no extension
interesting_extensions.append('.txt')
...
file_list = []
file_list.append('passwords')
file_list.append('passwd')
...

for user in users_result:
    home = users_result[user]['home']
    for interesting_file in file_list:
        for extension in interesting_extensions:
            file_fp = home + interesting_file + extension
            files_to_read.append( file_fp )
```

Demo “interesting_files”

Treasure hunt

RAPID7

Payloads are integrated into the framework

- Payloads can **take decisions based on facts that were saved to the knowledge base during the scan**:
  - Identified vulnerabilities
  - Remote Web server type (Apache, IIS, etc.)
  - Remote operating system
  - Found URLs

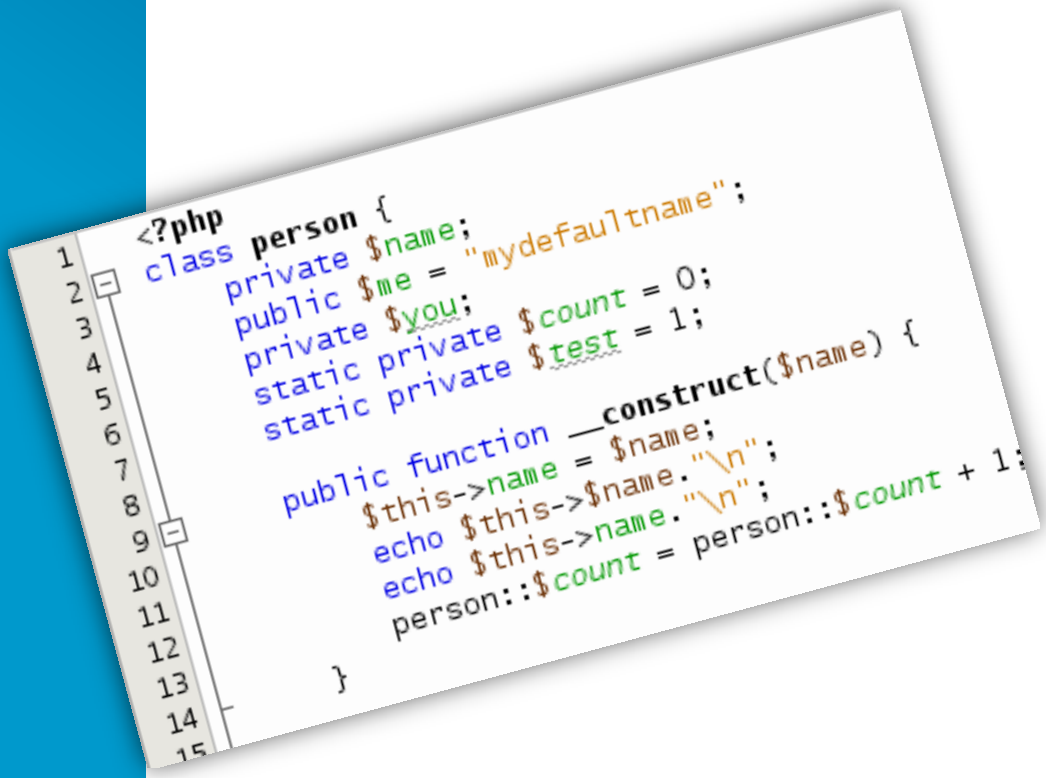
- This is **one of the biggest advantages** of having everything integrated into w3af!

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 “get_source_code”

w3af integration



RAPID7

w000t!

We have the application's source code, what now?

Integration with Static Code Analysis tools

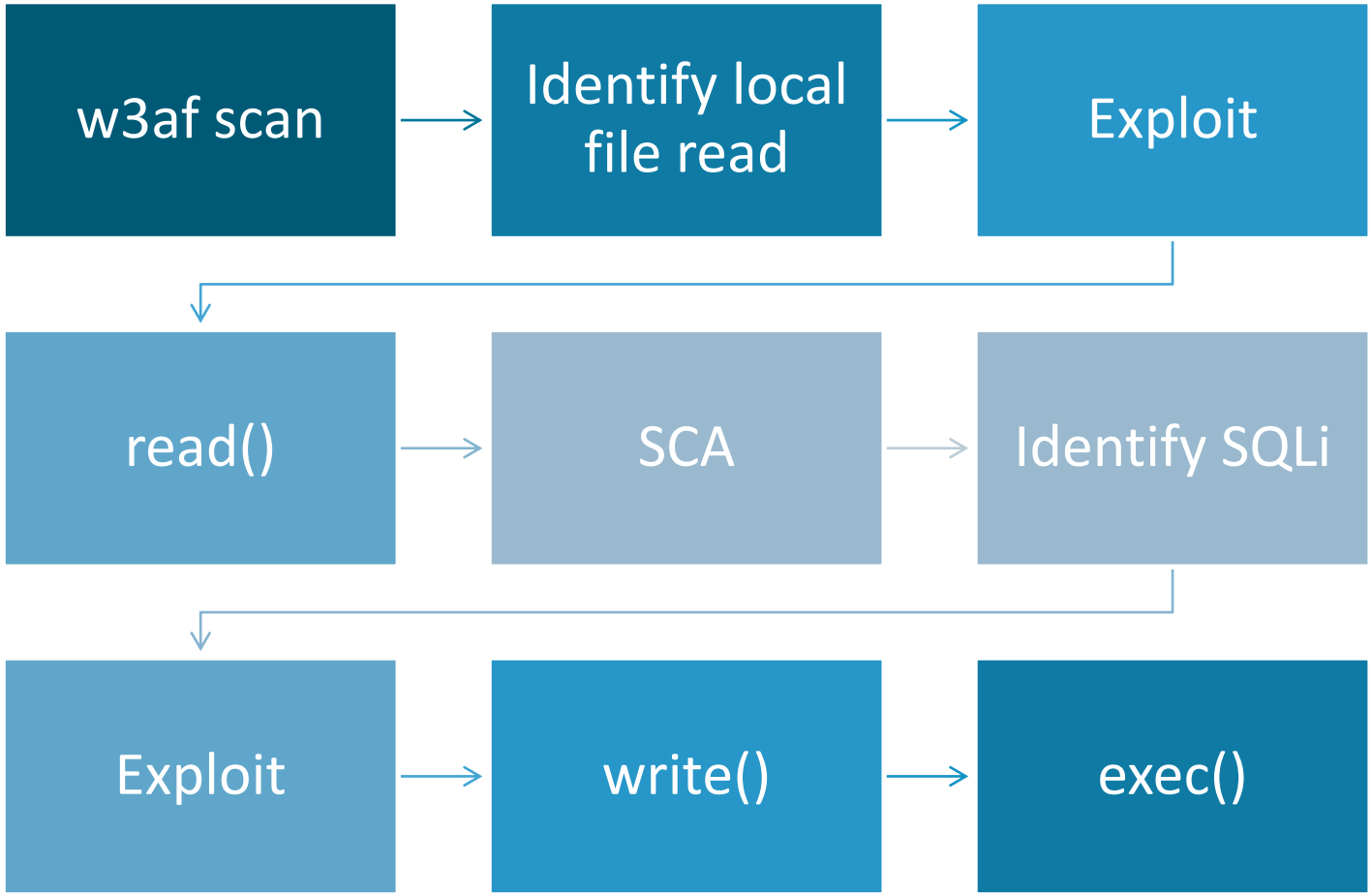
- Web application payloads can **easily integrate with other tools**. They are developed in Python, so everything is possible :)

- Our first stab at this problem was to integrate Pixy as a payload. The worse thing was that **it did not return the information we needed.**

- Together with Javier Andalia from Rapid7 **we've developed a PHP Static Code Analyzer** as a PoC to show that it is possible to **combine these two technologies:**
  - **Black-Box scanning**
  - **Static Code Analysis**

Integration with Static Code Analysis tools

- This is how we're **integrating our SCA tool into w3af:**



RAPID7

Static Code Analysis characteristics

- **Based on phply**, a PHP parser implemented in **PLY** (Python Lex-Yacc)

- **Identifies the following vulnerabilities:**
  - **SQL Injection**
  - **OS Commanding**
  - Arbitrary file read
  - Remote file inclusion
  - eval() vulnerabilities

- **Taint analysis**

Demo Static Code Analyzer

A step closer to retirement

RAPID7

Static Code Analysis with Taint Analysis

- This SCA was a **PoC developed over two weeks, it lacks many important functions** such as:
  - **Support** for require_once() , require(), include_once(), include()
  - Better support for loops and if statements
  - Classes, methods and attributes
  - Detection for all vulnerabilities
- **Interested in extending this section of w3af?** Contact me!

Available payloads and their main focus

Payloads with exec()

That was easy!

RAPID7

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:
  - msf_linux_x86_meterpreter_reverse
  - msf_windows_meterpreter_reverse_tcp
  - msf_windows_vncinject_reverse
  - w3af_agent
    - Allows us to **route traffic through the compromised host** without any effort

Conclusions and pending work

- Develop more MS **Windows** payloads
- **Take actions** based on payload results:
  - Launch a new scan against a particular resource
  - Exploit vulnerabilities using the increased knowledge obtained by w3af's payloads
- Our goal is to make this the **standard for automatized post-explotation** of Web application vulnerabilities.

Sharing your ideas and knowledge is easy!

- **Got an idea? Share it in our mailing list!**

  http://www.w3af.org/mailing-list.php

- **Want to read the 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/

Time for your questions!

- Andrés Riancho
- **Director of Web Security**
- General Manager of Rapid7's **Web Application Center of Excellence in Buenos Aires**

- andres_riancho@rapid7.com
- Follow me on Twitter **@w3af**

RAPID7

Thank you!

Web Application Center of Excellence,
Buenos Aires, Argentina