Embedding Defense in Server-Side Applications

Fernando Arnaboldi
Senior Security Consultant

IOActive
Hardware | Software | Wetware
SECURITY SERVICES

Why Am I talking About This?

- I write code
- I audit code
- I embedded defense in code

IOActive™

What is This Talk About

- Embedded defense specification for server-side applications
- Sample implementations in Java, .NET, PHP and Python
- Talk about how attackers reveal themselves
- How applications can recognize the attacks
- Show how applications can react

IOActive™

Intro

- There are two problems associated to application's security:
  - Developing secure code
  - Implementing a defense layer

- The proposed solution:
  - Embed the defense with the code

The First Problem:

Developing Secure Code

IOActive™

Developing Secure Code

- Applications try to rely in secure coding guidelines to create secure applications.
- That is not enough, there are always hidden bugs in the code.

IOActive™

Developing Secure Code (cont.)

- The default behavior of applications that deal with insecure requests is to just drop them.
- When the applications do not detect the attacks, they start leaking information.

Developing Secure Code (cont.)

- Let’s take for example one of the most secure codes. CERT is an authoritative organization formed by DARPA dedicated to improving the security and resilience of computer systems, which also define secure coding guidelines.

- What if CERT secure coding guidelines contain vulnerabilities?

Secure Compliant Example #1 from CERT

- "*This compliant solution implements the policy that only files that live in c:\homepath may be opened by the user and that the user is not allowed to discover anything about files outside this directory*"

Compliant Solution (Security Policy)

This compliant solution implements the policy that only files that live in `c:\homepath` may be opened by the user and that the user is not allowed to discover anything about files outside this directory. The solution issues a terse error message when the file cannot be opened or the file does not live in the proper directory. Any information about files outside `c:\homepath` is concealed.

The compliant solution also uses the `File.getCanonicalFile()` method to canonicalize the file to simplify subsequent path name comparisons (see FIO16-J. Canonicalize path names before validating them for more information).

```
class ExceptionExample {
  public static void main(String[] args) {

    File file = null;
    try {
      file = new File(System.getenv("APPDATA") +
             args[0]).getCanonicalFile();
      if (!file.getPath().startsWith("c:\\homepath")) {
        System.out.println("Invalid file");
        return;
      }
    } catch (IOException x) {
      System.out.println("Invalid file");
      return;
    }

    try {
      FileInputStream fis = new FileInputStream(file);
    } catch (FileNotFoundException x) {
      System.out.println("Invalid file");
      return;
    }
  }
}
```


IOActive™

Secure Compliant Example #1 (cont.)

Demo CERT #1

IOActive™

Secure Compliant Example #1 (cont.)

- *“only files that live in c:\homepath may be opened by the user”*

```
file = new File(System.getenv("APPDATA") +
       args[0]).getCanonicalFile();
if (!file.getPath().startsWith("c:\\homepath")) {
  System.out.println("Invalid file");
  return;
```

- **Fail #1: What about C:\\homepathfail?**

IOActive

Secure Compliant Example #1 (cont.)

```
file = new File(System.getenv("APPDATA") +
        args[0]).getCanonicalFile();
```

```
$ java ExceptionExample
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
        at ExceptionExample.main(ExceptionExample.java:11)
```

- **Fail #2: Did not handle all exceptions**
- **Fail #3: Did not validate method arguments**

IOActive™

Secure Compliant Example #1 (cont.)

```
if (!file.getPath().startsWith("c:\\homepath")) {
  System.out.println("Invalid file");
  return;
}
```

- Fail #4: They may not log the error
- Fail #5: They know that someone is attacking the filesystem and no real actions are being taken

IOActive™

Define the application minimum security level

IOActive™

Define the application minimum security level

- Fail 1 shows that an attacker may be able to access information from other directories.

- Do you want to know when attackers exfiltrate information?

IOActive™

Define the application minimum security level (cont.)

- Fail 2, 3 and 4 provide different forms of unhandled exceptions.
- Do you want to know if someone is triggering expected or unexpected exceptions?

IOActive™

Define the application minimum security level (cont.)

- Fail 5 shows that the developer was aware that someone may try to attack the application.

- Do you want to do something when you're aware of an attack?

IOActive™

The Second Problem:

Implementing a Defense Layer

IOActive™

Previous Work on Embedded Defense

- OWASP Java AppSensor is the most mature embedded defense
- One programming language
- It doesn't consider all the attack vectors

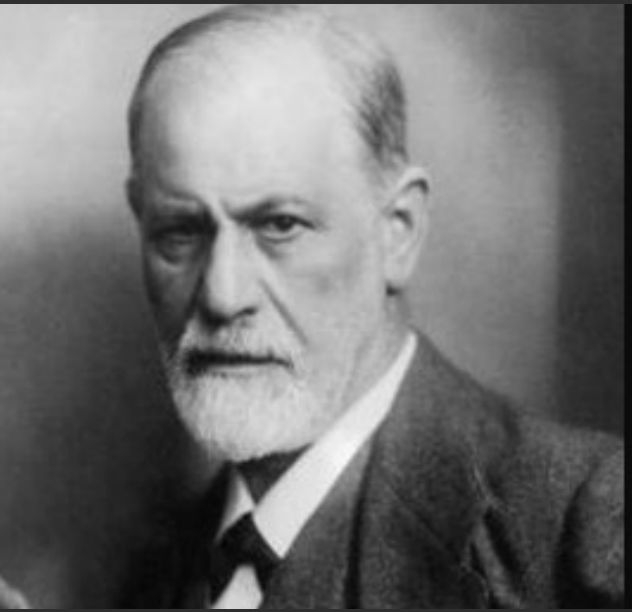
IOActive™

Problems of Using a Detached Defense

- They may introduce new vulnerabilities.
- Application's source code always contains the different places to embed the defense.
- Certain defense solutions work by analyzing the logs after the incident happened, with no real time protection.
- Detached defenses require additional permissions to be installed and can be unique to certain environments.

IOActive™

Vulnerabilities are Valuable



Out of your vulnerabilities will come your strength.

— Sigmund Freud —

IOActive™

The Proposed Solution:

Integrate the Defense within the Code

IOActive™

Proposed Solution

Can you imagine what I would do if
I could do all I can?

Proposed Solution

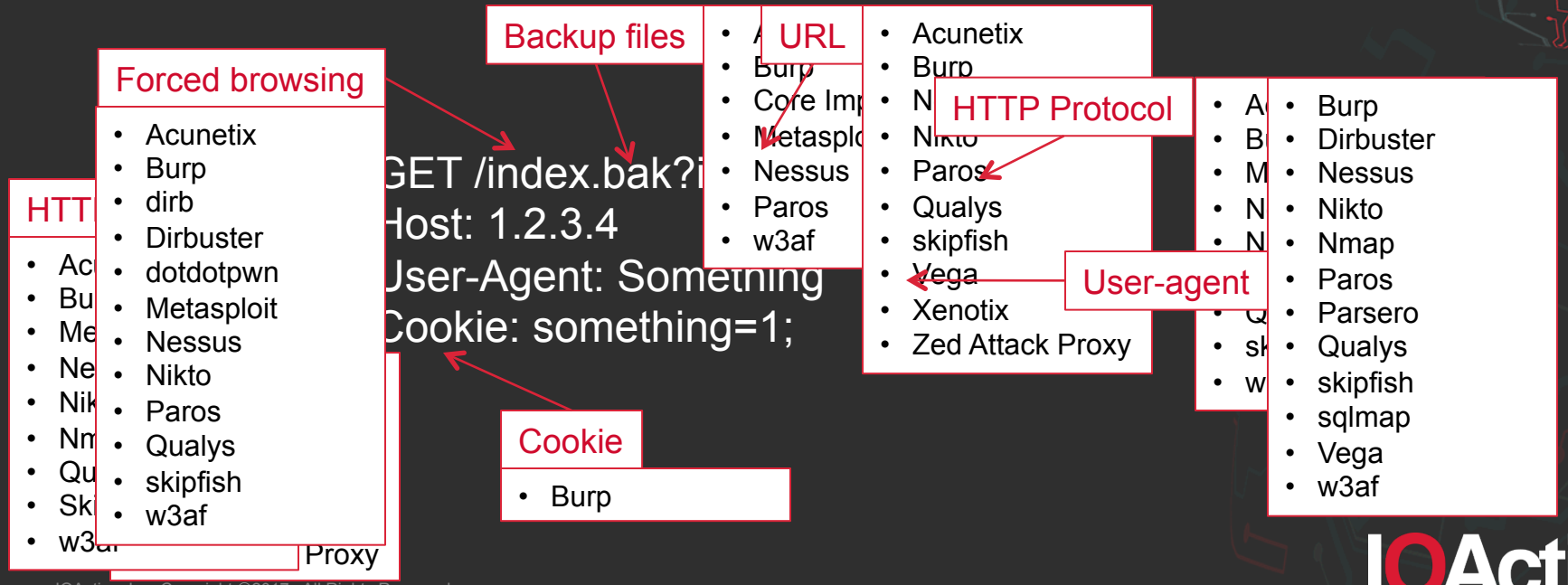
- A comprehensive open source specification of defensive measures
- Integrate the defense in three places of your code:
  1. Pre-execution: before the functionality is executed
  2. Execution: while the functionality is executed
  3. Post-execution: after the functionality was executed

1. Pre-execution Controls

IOActive™

Pre-execution Controls: Tools Disclosure

- Tools can be identified by multiple factors. Consider an HTTP request:



IOActive

Pre-execution Controls: Tools Disclosure (cont.)

Tool	User-Agent	-based Connection	Forced Browsing	Retrieves Backup Files	Retrieves robots.txt Files	URL Disclosure	HTTP Verb
Acunetix WVS	no	no	yes	yes	yes	yes	yes
Burp Suite	yes	no	yes	yes	yes	yes	yes
Core Impact	n/a	n/a	n/a	yes	yes	n/a	n/a
dirb	no	no	yes	no	no	no	no
Dirbuster	yes	no	yes	no	no	no	no
dotdotpwn	no	no	yes	no	no	no	no
Metasploit (wmap)	no	no	yes	yes	no	no	yes
Nessus	yes	yes	yes	yes	yes	yes	yes
Nikto	yes	yes	yes	no	yes	yes	yes
Nmap	yes	no	no	no	no	no	yes
Paros	yes	no	yes	yes	no	yes	no
Parsero	yes	no	no	no	yes	no	no
QualysGuard	yes	yes	yes	no	no	yes	yes
skipfish	yes	no	yes	no	no	yes	yes
sqlmap	yes	no	no	no	no	no	no
Vega	yes	no	no	no	no	yes	no
w3af	yes	yes	yes	yes	yes	n/a	yes
WebScarab	no	yes	no	no	no	no	no
Xenotix Framework	no	no	no	no	no	yes	no
Zed Attack Proxy	no	yes	no	no	no	yes	no

IOActive

Pre-execution Control: Diversion/Trap Example

- Developers can set up diversions for attackers:

```
<form method="post">
  Username: <input id="user" type="text" name="user" ><br/>
  Password: <input id="pass" type="password" name="pass">
  <input type="hidden" name="passkey" value="check">
</form>
```

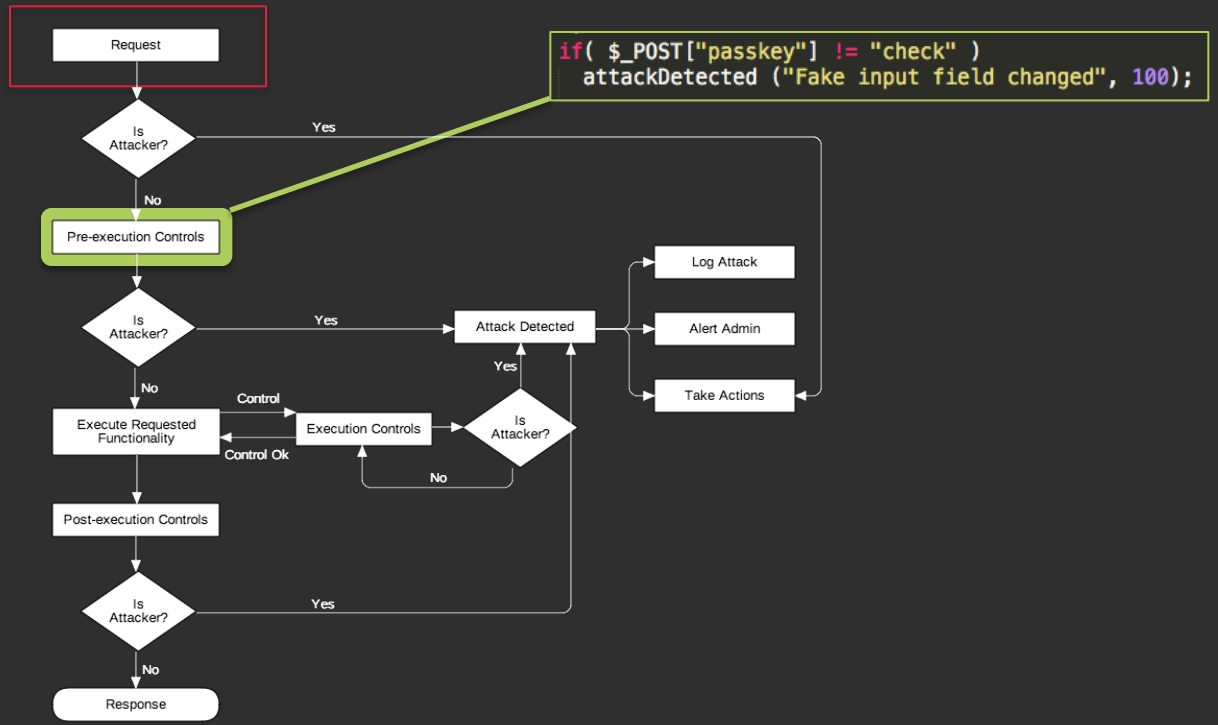
- Or transform diversions into traps:

```
if( $_POST["passkey"] != "check" )
  attackDetected ("Fake input field changed", 100);
```


IOActive

Pre-execution Control: Trap Example (cont.)

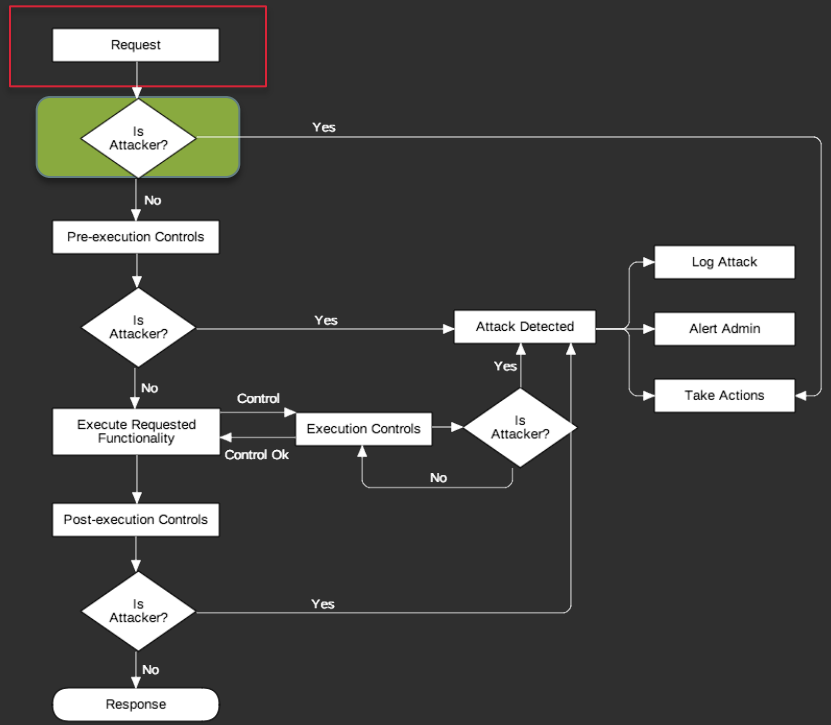
Checking `passkey` with the value `fuzzing1'`



IOActive™

Pre-execution Control: Trap Example (cont.)

Checking `passkey` with the value `fuzzing2'`



IOActive™

Why Use Pre-Execution Controls?

- Attacker tools and techniques can be identified before reaching the main code
- Diversions may entice attackers to focus on incorrect areas
- Traps can expose attackers

IOActive™

2. Execution Controls

IOActive™

Execution Controls

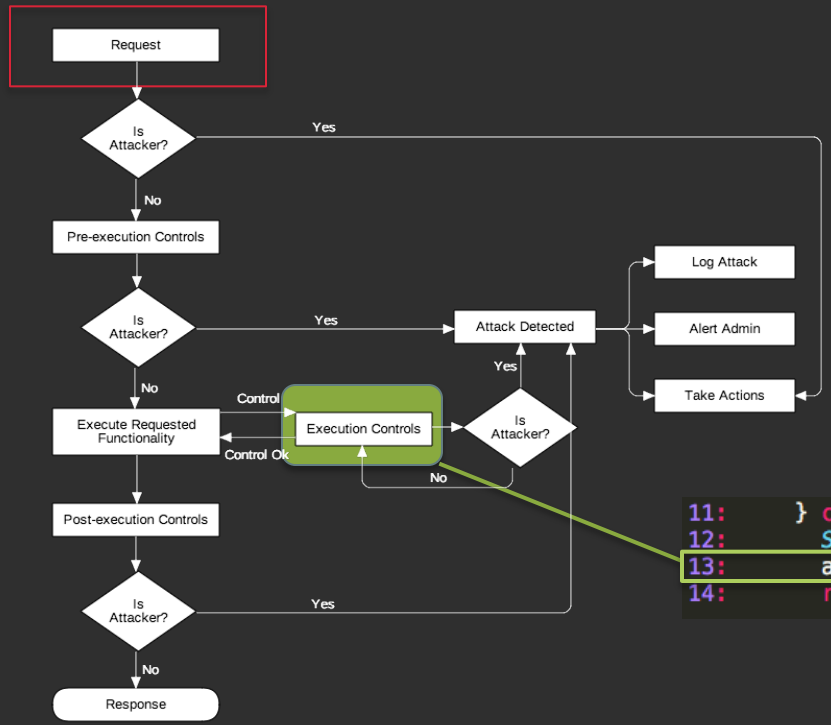
- Different protections throughout the code can be embedded:
  - Check if they are trying to perform a man-in-the-middle (MiTM) attack
  - Check if they are trying to exploit a path traversal
  - Check if they are trying to bypass an anti cross-site request forgery (CSRF)
  - Check if they are triggering expected exceptions
  - Check if they are triggering uncaught exceptions
  - Check if they are trying passwords in a loop
  - And so on…

Execution Controls

- Remember the CERT example for “homepath”?
- The most serious vuln was related to an unlikely path traversal.
- We will see later how a post-execution control may stop that attack after the execution.
- But right now we know there is an attack going in place. What can we do?

Execution Control: Path Traversal Example

Checking a malicious string “C:\\homepath\\..\\attacker_Controlled”

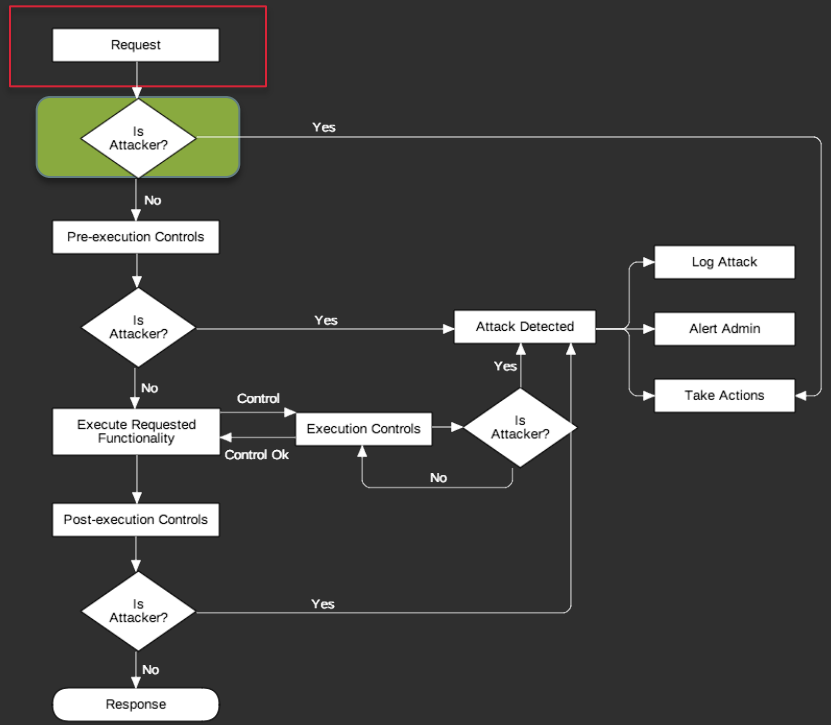



```
11:     } catch (IOException x) {
12:       System.out.println("Invalid file");
13:       attackDetected("Path traversal detected", 100);
14:       return;
```


IOActive™

Execution Control: Path Traversal Example (cont.)

Checking a malicious string "C:\\homepath\\..\\attacker_Controlled"



IOActive™

Why Use Execution Controls?

- Certain controls can only be applied when being executed
- For example, the expected exception of the C:\homepath
- Or the unexpected exception of that program when executed without parameters

IOActive™

3. Post-execution Controls

IOActive™

Post-execution Controls

- Sensitive information: what information attackers may want to exfiltrate? (i.e. passwords, filenames)
- Time information: gain knowledge by timing (i.e. authentication, big requests)
- Size information: tools analyze the behavior based on contents and size
- Environment information: web servers may try to provide information (time)

Secure Compliant Example #2 from CERT

- *"To prevent vulnerabilities, a program must operate only on files in secure directories. […] file links can be swapped out and may not always point to the intended location. As a result, file links in shared directories are untrusted and should not be operated on"*

```
try {
  if (Files.isSymbolicLink(partialPath)) {
    if (!isInSecureDir(Files.readSymbolicLink(partialPath),)) {
                        user, symlinkDepth - 1)
      // Symbolic link, linked-to dir not secure
      return false;
    }
```


IOActive™

Secure Compliant Example #2 (cont.)

Demo CERT #2

IOActive™

Secure Compliant Example #2 (cont.)

- 1) Create a symlink and a hard link:

```
$ ln -s /tmp/extract_data /Users/test/symlink
$ ln /tmp/extract_data /Users/test/hardlink
$ ls -l /Users/test/symlink /Users/test/hardlink
lrwxr-xr-x  1 test  staff  17 Mar  8 16:33 /Users/test/symlink -> /tmp/extract_data
-rw-r--r--  2 test  wheel   0 Mar  8 16:32 /Users/test/hardlink
```

- 2) Check the symlink

```
$ java isInSecureDir /Users/test/symlink
Accessing: /tmp/extract_data
File not in secure directory
```

- 3) Check the hard link:

```
$ java isInSecureDir /Users/test/hardlink
Accessing: /Users/test/hardlink
$
```

IOActive™

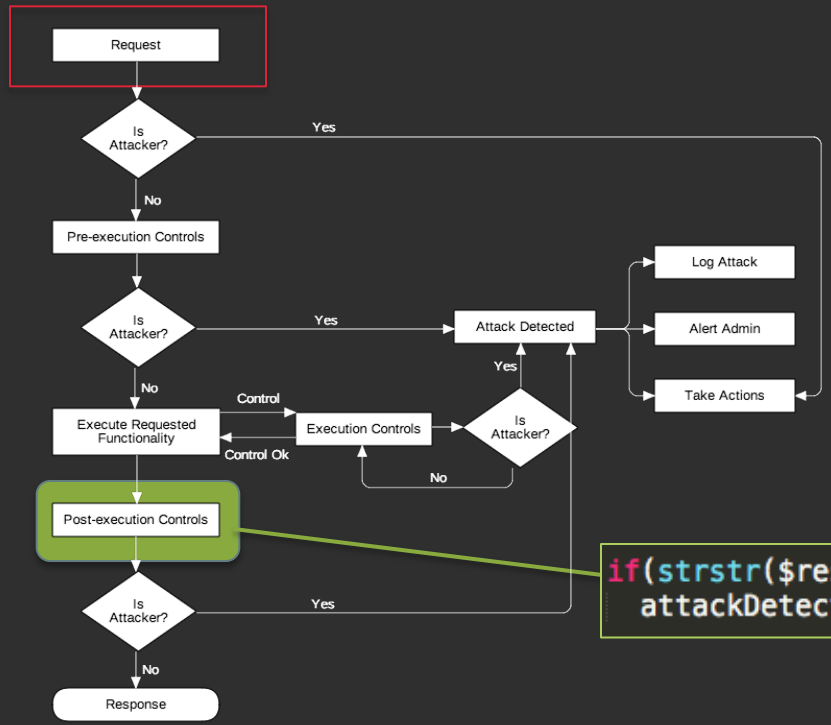
Post-Execution Control: What Now?

- Assuming the previous control was in place, the attacker may be ready to exfiltrate information.
- There are certain pieces of information that are valuable for the application or attackers

IOActive™

Post-execution Control: Exfiltration (cont.)

Checking if a special username is being exfiltrated




```
if(strstr($response, "secretAdmin")) // fake admin user
  attackDetected("Passwords leaked", 100);
```


IOActive™

Why Use Post-Execution Controls?

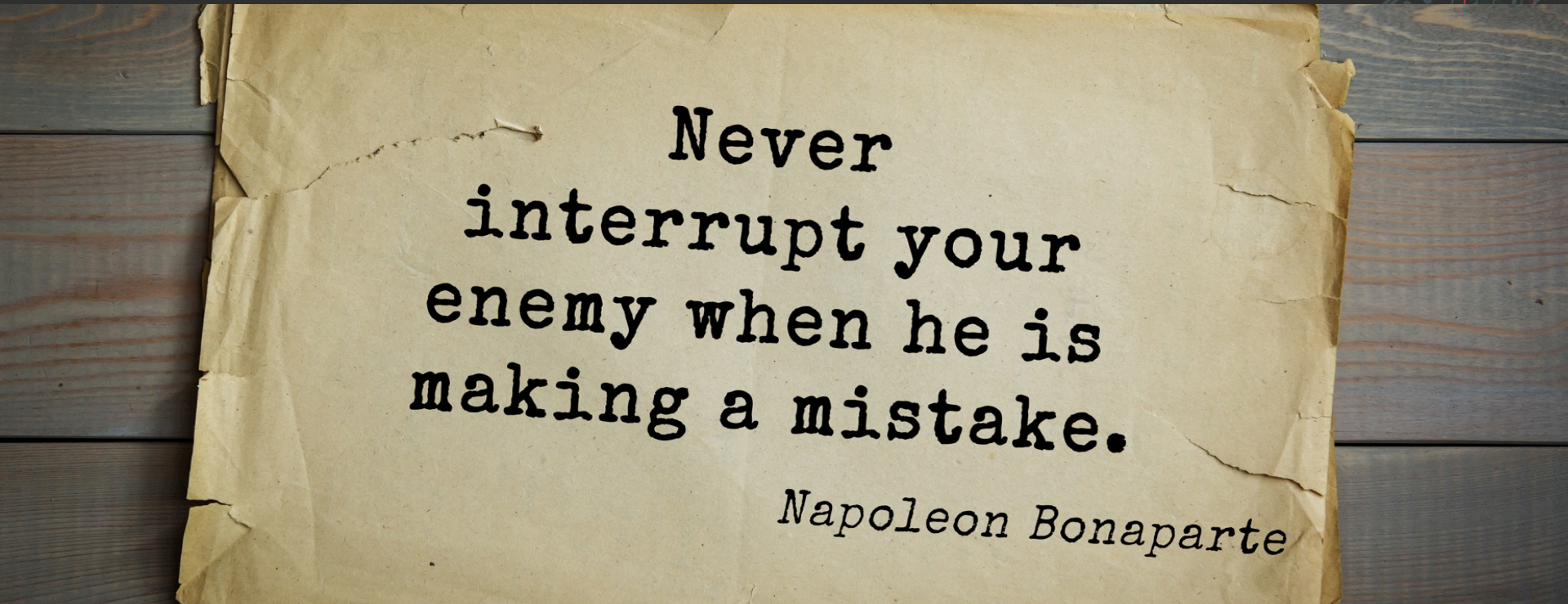
- Avoid sharing sensitive information
- Avoid attackers abusing timing or size related attacks
- Avoid the environment from providing extra pieces of data

IOActive™

Take Actions

IOActive™

Take Actions



Take Actions (cont'd)

- Log and share your knowledge
- Alert the administrator or security personnel

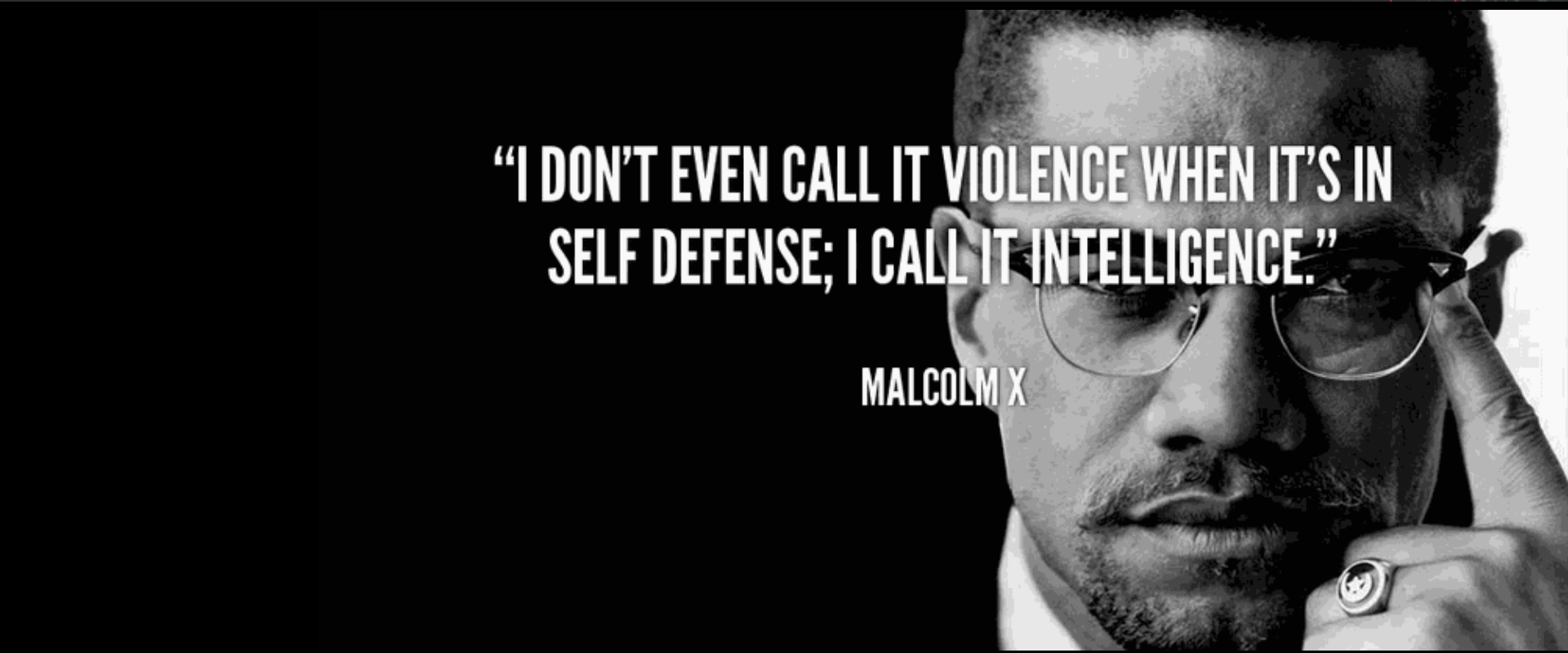
IOActive™

Take Actions (cont.)

- React to the attacker, do not ignore it:
  - Distract: introduce fake functionalities or potential vulnerabilities
  - Slow down: make the process slower
  - Stop: avoid dealing with them and eventually disconnect the application
  - Bait: give something to the attacker that may help to prosecute them (i.e. a bank account or a credit card information)
- Legal actions

IOActive™

Take Actions (cont.)



Take Actions (cont.)

- Implement it in your language of choice:

Java for Jetty

```
29  public class Defense {
30
31          public static final int OK = 1;
32          public static final int ERROR = 0;
33          public static final int ATTACK = -1;
34
35
36
37
38
39
40
41
42
43
```

C# for IIS

```
11  namespace TestDefense.Controllers
12  {
13      public class HomeController : Controller
14      {
15
16
17
18
19
20
21
22
23
24
25
```

PHP for Apach

```
2   class defense {
3       const OK = 1;
4       const ERROR = 0;
5       const ATTACK = -1;
6       const BAN = 100;
7       const DEBUG = true;
8       const DB = "attackers.db";
9       const NEWLINE = "\n";
10
11      // Check that the user is using the correct HTTP meth
12      function checkHttpMethod($method="") {
13          $attack = "Incorrect HTTP method";
```

Python for Django

```
2   import os.path
3   import datetime
4   import time
5   from django.contrib.sessions.middleware import SessionMiddleware
6   from django.http import HttpResponsePermanentRedirect
7   import os.path
8
9
10  class handling_middleware:
11      OK = 1
12      ERROR = 0
13      ATTACK = -1
```


IOActive™

Take Actions (cont.)

Testing Self Defense

IOActive™

Conclusions

- Do not ignore attacks: do something.
- Suggest defense mechanisms when doing source code analysis.
- Be dynamic and share your attack knowledge.

IOActive™

Questions?

IOActive™

Thank You

IOActive™