WHITE PAPER

Embedding Defense in Server-Side Applications

Draft Version

Fernando Arnaboldi

IOActive Senior Security Consultant

Abstract

Applications often rely on secure development practices and third-party defense mechanisms for protection. Whenever an application receives malicious payloads they are either dropped or executed by the affected application. Ignoring these situations aid attackers in performing deep analysis of applications until they are able to exploit existing flaws.

Standards, libraries and third-party defense systems developed to secure applications introduce opportunities for attackers. While some protections have already been implemented in applications and web firewalls, there is a whole spectrum of techniques not being analyzed. This research details how server-side applications can incorporate an extensive layer of defense to detect and protect against attackers.

Defense mechanisms will be released in four different languages: .NET, Java, PHP and Python. Throughout the presentation, undisclosed vulnerabilities from secure coding guidelines will be used to exemplify. By implementing the defenses laid out in this paper, attackers may unwittingly become the victims.

IOActive™
Hardware | Software | Wetware
SECURITY SERVICES

© IOActive, Inc. All Rights Reserved

Contents

© IOActive, Inc. All Rights Reserved.

Introduction

Applications will always have security flaws, which is why we rely on multiple layers of defense. This specification is intended to serve as a guide to rethink how applications defense should be approached. The goal is to introduce defensive methods that cause attackers to cease or take incorrect actions based on false premises.

Certain defense techniques presented in here are already present in solutions like intrusion protection systems (IPS) and web application firewalls (WAF). These products are a valuable resource when basic, off-the-shelf protection is required or the application source code is not available. However, implementations in native code provide a greater degree of defense that cannot be achieved using third-party solutions. It is worth noting the OWASP project to self-defend Java web applications[1] that analyzes several of the techniques here described. Nevertheless, this document aims to:

- Disclose a comprehensive list of current and new defensive techniques:
  Different defensive techniques have been disclosed in books, papers and tools. This document collects those techniques and presents new defensive opportunities to fill the gap of opportunities present for attackers.

- Disclose how to identify attack tools before they access the functionalities.
  The tools used by attackers disclose themselves in several ways. There are multiple features of a request that can disclose it is not from a normal user; they abuse security flaws in ways that serve to detect which type of tool the attacker is using and developers can prepare diversions and traps in advance to be triggered automatically.

- Disclose how to detect attacker techniques within code.
  Certain techniques can be spotted throughout the code. Developers may know in advance that certain conditions will only be triggered by attackers, and they can also be prepared for certain unexpected scenarios.

- Provide a new defensive approach.
  Defense is normally about blocking malicious IP addresses associated to attackers; however, an attacker's focus can be diverted or modified. Sometimes certain functionalities may be presented to attackers only to better prosecute them if they are triggered.

- Provide these protections for multiple programming languages.
  Functionalities to reduce the effectiveness of attackers and expose their actions will be explained throughout this document using pseudo code, and working proof of concept pieces of code will be released for four programming languages: Java, Python, .NET and PHP.

[1] OWASP AppSensor Project (https://www.owasp.org/index.php/OWASP_AppSensor_Project)

Setting Up a Strong Defense

Application defense applies to any type of software; however, server-side applications are more attractive for a couple of reasons. First, the code is normally never exposed to the attackers, so they are forced to interact from a black box perspective without knowledge on how the defenses were deployed. Secondly, server-side applications control what will happen until they decide to interact or respond to the user.

The application must implement a minimum set of functionalities before an attack in order to treat real users and attackers differently. Sometimes, one request will be enough to target the actor as an attacker; other times, an accumulation of malicious actions will be required based on a scoring system.

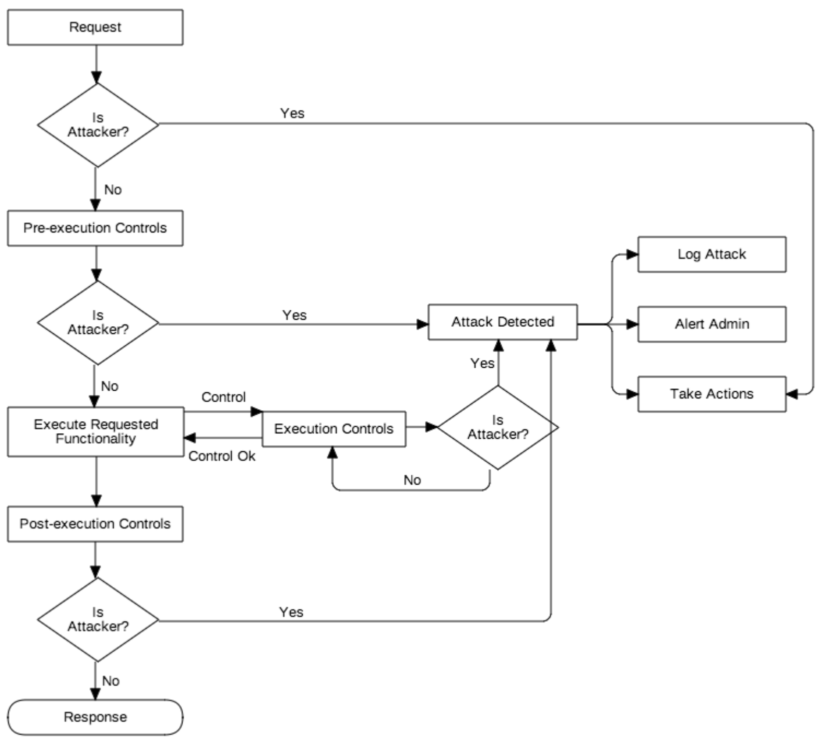

Figure 1: Defense flow chart

As illustrated in Figure 1, defensive controls should exist throughout the application:

- When a request is received, the first control is to check if the session (whether the IP address, the user, or the cookie) appears to be from an attacker.
- Prior to accessing the application's functionality, requests should be analyzed in the "Pre-execution Controls" to detect attack tools and malicious actions.
- Once the session and the request are authorized, the application's functionality should include "Execution Controls" to detect specific attacks affecting the code.
- Finally, before sending a response to the user, the contents should be analyzed in the "Post-execution Controls" to detect and prevent malicious scenarios.

Whenever a malicious action is detected, the type of attack, the associated score of the attack, and the session information should be logged. If the attacker's actions rise above a certain level, the IT security team should be alerted and specific actions should be taken.

Following different forms of controls for web applications will be explored.

Pre-execution Controls

Certain requests provide information that an attacker is trying to abuse a resource. This will either happen because of the characteristics of the request, or because they were caught in a trap.

Characteristics

Certain characteristics reveal if a request is from a legitimate user or part of an attack. The information contained in the `HTTP` protocol, `URL`, `User-Agent`, and `Host` can provide information about the type of tactics being used.

Malicious HTTP Method

Web applications commonly rely in a set of `HTTP` methods: `GET`, `POST`, etc. Some tools will try to abuse non-existent `HTTP` methods (Acunetix uses the `HTTP` method "`ACUNETIX`", w3af uses "`ARGENTINA`", Qualys uses "`ABCD`", etc.), while others try potentially vulnerable methods (the `TRACE` or `TRACK` methods are a classic tested by Burp, Nessus, Nikto, and others). Certain methods may be tested even when they are not being used by your application (like `PROPFIND`, `DEBUG`, etc.).

Constructing a whitelist of valid and accepted methods allows the application to distinguish between valid users and attackers:

```
$foundValidHttpMethod = false;
$results = $db->query("SELECT method FROM validHttpMethods");
while ($row = $results->fetchArray())
  if($REQUEST_METHOD == $row["validMethod"])
    $foundValidHttpMethod = true;
    break;
if(!$foundValidHttpMethod)
  attackDetected("Blacklisted HTTP method", 25);
```

Code 1: Pseudocode from the defense class to detect incorrect HTTP methods

A precise check of the `HTTP` method should be made for each function. This is discussed in greater detail in "Incorrect HTTP Method".

Malicious URL and/or Value

The name of the tool sometimes appears as part of the URL, either as the filename or the value for some of the parameters (like Acunetix, Burp, Nessus, Nikto, Nmap, Paros, Qualys, Vega, and Zed attack proxy):

```
$results = $db->query("SELECT denyString FROM denyUrlString");
while ($row = $results->fetchArray())
  if(strstr($REQUEST_URI, $row["denyString"]))
    attackDetected("Vulnerability scanner in URL", 10);
```

Code 2: Pseudocode from the defense class to detect vulnerability scanners in the URL

If the application does not properly perform input sanitization, then malicious payloads can be included in this check from a blacklist perspective (like IPS or WAF do).

Invalid HTTP Protocol Version

Certain tools like Nessus, Qualys, and w3af try to use HTTP protocols that are intentionally not supported or even malformed (like "`HTTP/rndmmtd`"). Since this value is fuzzed to look for errors, it is better to whitelist accepted values:

```
if($SERVER_PROTOCOL != "HTTP/1.1") {
  attackDetected("Incorrect HTTP Version", 100);
```

Code 3: Pseudocode from the defense class to detect incorrect HTTP protocols

Incorrect Hostname (or IP-based Connection)

When the web server receives requests using only the IP address without referencing the hostname (either using `HTTP/1.0` without the `Host` or by only referring to the IP address in the `Host` with other HTTP protocols), the request could come from an attack tool that is only using the IP address to access the application (and to eventually speed up the process):

```
if($HTTP_HOST != "www.example.com") {
  $this->attackDetected("Incorrect hostname", 100);
```

Code 4: Pseudocode from the defense class to detect an IP-based connection

Forced Browsing

Forced browsing occurs when a user tries to access a resource that either does not exist or for which they do not have the required permissions[2].

There are five different ways to check for forced browsing on the server side:

1. Invalid URI:
   Whenever tools that perform path traversal (dotdotpwn, Burp, etc.) try to connect to invalid URI resources (because they may be trying to access "`/../something`"), they will hit an `HTTP 400` error code because the directory is invalid. Simply check for that on the web server config:

```
ErrorDocument 400 /invalidURI
```

Code 5: Configuration for an HTTP 400 error code in Apache

[2] CWE-425: Direct Request ('Forced Browsing') (https://cwe.mitre.org/data/definitions/425.html)

Then, add the log of the potential attack in the `invalidURI` webpage:

```
1: attackDetected("Invalid URI (potential path traversal)", 20);
```

Code 6: Call attackDetected() when the file "invalidURI" is accessed

2. Non-existent resource:
   It is possible that a legitimate user may try to access a non-existent resource for several reasons:

   - The functionality was present in the application; it was stored in the browser's cache and has been removed server side.
   - The functionality was present in the application; a web crawler identifies it and now it has been removed server side.
   - The functionality was dynamically present in the user session.

   Having a control here will prevent brute-force attacks on files and directories (when using DirBuster, DIRB, etc.). Nevertheless, it is a good idea to give this type of condition a low score to prevent normal users from hitting this trap by mistake.

   The web server could be set up to show a specific webpage whenever someone hits an `HTTP 404`:

```
ErrorDocument 404 /notfound
```

Code 7: Configuration for an HTTP 400 error code in Apache

The first action of the `notfound` web page should be to log the user's action.

```
1: attackDetected("Non-existing resource", 5);
```

Code 8: Call attackDetected() when the file "notFound" is accessed

3. Backups related to existent files:
   Multiple tools try to expose sensitive information stored in backup files[3]. For instance, whenever certain tools find a `menu.php` file in the web server, they will try to access the associated backup files: `menu.bak`, `menu.old`, etc. This is the same as the previous condition (accessing a non-existent resource), but with a higher score if they reference backup files: these files were never presented to users.
   This control can either be set up as a general rule (mark all URLs containing the "`.bak`" string as an attack) or as a specific trap for a certain webpage:

[3] CWE-530: Exposure of Backup File to an Unauthorized Control Sphere (https://cwe.mitre.org/data/definitions/530.html)

```
1: $results = $db->query("SELECT extension FROM denyExtension");
2: while ($row = $results->fetchArray()) {
3:   if($extension == $row["extension"]) {
4:     attackDetected("Non existing backup file", 100);
```

Code 9: Pseudocode from the defense class to check if the file extension is part of an attack

Line 3 will check if the user was trying to access a forbidden file extension.

4. Non-authenticated user accessing an authenticated resource:
   This may happen legitimately if a user tries to access functionalities using an expired cookie. It is not legitimate if there is no authentication or cookie in place and the user is just trying to access the privileged resource:

```
if (!user->isLogged())
  attackDetected("Existing resource accessed by a non-authenticated user", 20);
```

Code 10: Call attackDetected() whenever a non-authenticated user is trying to access an authenticated resource

5. Authenticated user accessing a privileged resource without permission:
   This is probably the most critical scenario. There is no gray area where this has happened by mistake. This should be flagged automatically as an attack and action taken:

```
if (user->isLogged() && !user->isAuthorized())
  attackDetected("Authenticated user without permission", 100);
```

Code 11: Call attackDetected() when a non-authorized user is trying to access an authorized resource

User-Agent

Even though the value of the User-Agent can be controlled by an attacker, these values are normally left intact or inadvertently disclosed. Attack tools may expose their names in the `User-Agent` (for example, sqlmap includes the string "sqlmap"). Other tools can reveal that they are not normal web browsers (for example, programming languages or command line tools expose their primitive names) and some try to disguise their actions behind what would be consider normal User-Agents. But even these normal User-Agents may expose unusual activities. WMAP, developed by Offensive Security for Metasploit, uses a `User-Agent` from Internet Explorer 6.0, which is unlikely to be supported by current web applications.

The following detects the direct disclosure of the tool names for Burp, Dirbuster, Nessus, Nikto, Nmap, Paros, Parsero, Qualys, sqlmap, Vega, and w3af:

```
$results = $db->query("SELECT useragent FROM denyUserAgent");
while ($row = $results->fetchArray())
  if(strstr($HTTP_USER_AGENT"], $row["useragent"]))
    attackDetected("Vulnerability scanner in user-agent", 100);
```

Code 12: Pseudocode from the defense class to detect vulnerability scanners with the User-Agent

Sometimes the `User-Agent` is changed during the user's session to introduce payloads in them or to disguise an attack. The initial value of the `User-Agent` should be set for the user's session, and if it changes, it should be detected:

```
if($USER_AGENT  != $userSession["user_agent"])
  attackDetected("User-agent changed during user session", 100);
```

Code 13: Pseudocode from the defense class to detect if User-Agent changed

Cookies

Cookies should be associated with the user’s IP address whenever they are set. It is understandable that a user’s IP address may change during a user session, but it will be unlikely that this happens more than a certain amount of times:

```
if($REMOTE_ADDR != $userSession["remote_addr"])
  attackDetected("The IP address of the user changed for the same cookie",
   25);
```

Code 14: Pseudocode from the defense class to detect if the original IP address changed

Traps (and Diversions)

Traps–also referred as honey traps–are functionalities in the application created to attract and detect malicious actions. This is a well-known area of application defense that is frequently exploited automatically by vulnerability analysis and penetration testing tools.

Some traps may also be set up as diversions: they will not trigger an alarm, but they may entice the attacker to spend time trying to solve situations that do not have real functionalities. Adding one line of code may be enough to accomplish this goal.

Traps and diversions can be associated with the user's session. By keeping an association of which traps are being shown to which users, the logs can further explain how the attacker performed a series of actions using different IP address and/or cookies.

Fake robots.txt Entry

You can set up a fake disallow entry in your robots.txt file that should never be accessed by web crawlers or common users. The following `URL /testing` will be marked to not be accessed in this `robots.txt` file, but it may be retrieved automatically by penetration testing tools (like Burp, Core Impact, Nessus, Nikto, Parsero, and w3af):

```
User-agent: *
Disallow: /testing
```

Code 15: Sample robots.txt file

The contents of the fake robots.txt entry must alert the type of request received:

```
attackDetected("Fake robots.txt entry access", 100);
```

Code 16: Call attackDetected() when the file "testing" is accessed

Fake Hidden URL

Users or web crawlers will not normally process URLs within comments. Nevertheless, automatic vulnerability analysis tools will try to access these hidden functionalities:

```
<!-- <a href="/oldadmin">Old Admin Interface</a> -->
```

Code 17: Hidden URL line to be implemented on any valid resource

© IOActive, Inc. All Rights Reserved.

Searching for this common weakness[4] is common among vulnerability analysis and penetration testing tools. Whenever you receive a request in the previously defined URL "`/oldadmin`" server side, it should be flagged as an attack:

```
attackDetected("Fake hidden URL access", 100);
```

Code 18: Call attackDetected() when the file "oldadmin" is accessed

Fake Cookie

Cookies can also be set up as traps for attackers. Modifying the name of the cookie or its value on any request could be used to detect an attack. If attackers see a cookie named "`admin`" with the value "`false`", they may be tempted to change it to "`true`". In the following example, a fake admin cookie is being defined for the user (line 4), if this value is ever changed (line 6), it will be flagged as an attack (line 7):

```
1: $cookie_name = "admin";
2: $cookie_value = "false";
3: if(!isset($cookie[$cookie_name])
4:   $cookie[$cookie_name] = $cookie_value; // set a fake cookie named "admin"
   with a "false" value
5:   $userSession["cookie_"+$cookie_name] = $cookie_value;
6: if($cookie[$cookie_name] != $userSession["cookie_"+$cookie_name])
7:   attackDetected("Fake cookie changed", 100);
```

Code 19: Pseudocode from the defense class to detect when fake cookies are modified

Fake Input Field

Parameters are the entry point for web applications and are normally brute-forced and fuzzed to exploit security issues. Tools may commonly try to exploit the name of the parameter and/or its value.

A fake input field can be set as part of the parameters on any form either as a trap or diversion:

```
<form method="post">
  Username: <input id="user" type="text" name="user" ><br/>
  Password: <input id="pass" type="password" name="pass">
  <?
  $randomdata = substr(randomString(),rand(0, 32));
  echo '<input type="hidden" name="passkey" value="'. $randomdata . '">';
  ?>
</form>
```

Code 20: Fake input field implemented on the login form

In this example the input field named "`passkey`" is useless, it only contains a random piece of data that is not used on the server side. However, using an appealing name in the authentication may cause attackers to spend time dealing with it. Another approach could be to set the value of "`passkey`" in the session and later check if it was modified:

[4] CWE-615: Information Exposure Through Comments (https://cwe.mitre.org/data/definitions/615.html)

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

Code 21: Pseudocode from the defense class to detect if the fake parameter was changed

Execution Controls

After a request is considered to be trustworthy, it will be processed by the application. The following controls can be embedded to detect and protect the application from unwanted or unexpected conditions. It should be noted that these controls can be set up in multiple places within the same functionality when abnormal behaviors are detected.

Checks

The first line of defense is when functionalities verify the verb and parameters received. They reveal if the request was properly crafted for the appropriate functionality:

Incorrect HTTP Method

HTTP verbs need to be set up according to the requirements of each functionality. If the application is expecting to receive a `POST` request, anything different may be flagged as suspicious. Additionally, attackers may try to force a `GET` method when they are trying to exploit cross-site scripting (XSS) or cross-site request forgery (CSRF) issues:

```
if($REQUEST_METHOD != "POST" )
  attackDetected("Incorrect HTTP verb", 100);
```

Code 22: Pseudocode from the defense class to check for the correct HTTP verb

Missing Parameter

If the application is missing any parameter[5] in the request, it may indicate that the request has been tampered with. This type of situation is more unlikely to occur when dealing with requests that do not rely in the `GET` method:

```
if(isset($_POST['id']))
  $id = $_POST['id'];
else
  attackDetected("Missing parameter id", 20);
```

Code 23: Call attackDetected() whenever a parameter is missing

Extra Parameter

If the functionality receive more parameters than expected[6], it may indicate that the request was altered (again, this will be more unusual for `POST` requests):

[5] CWE-236: Improper Handling of Undefined Parameters (https://cwe.mitre.org/data/definitions/236.html)
[6] CWE-235: Improper Handling of Extra Parameters (https://cwe.mitre.org/data/definitions/235.html)

```
if(count($_POST)!=1) // only the parameter 'id' is expected in here
  attackDetected("Extra parameters", 20);
```

Code 24: Call attackDetected() whenever there are more parameters per form than expected

Unexpected Values

Applications are expected to perform proper data validation on user input. Whenever validation is performed on the client side to improve the user experience, the same controls must be enforced on the server side. If any difference is detected on the server side from the expected values, an attack should be flagged because someone bypassed the client-side controls. Consider the following parameter `id` that is supposed to be a number:

```
if(!is_numeric($id))
  attackDetected("Unexpected value", 100);
```

Code 25: Call attackDetected() whenever unexpected values are detected on the server side

Man-in-the-middle Check

Certain client-server applications may be capable of verifying the authenticity of the communications being performed to avoid man-in-the-middle (MiTM) attacks. For example, a server-side application may try to perform a SSH2 connection using a certificate to retrieve information. The moment that the certificate for that connection cannot be verified as authentic, it should trigger an alarm that someone may be trying to perform a MiTM attack:

```
$connection = ssh2_connect('shell.example.com', 22, array('hostkey'=>'ssh-
 rsa'));

if (ssh2_auth_pubkey_file($connection, 'username',
                          '/home/username/.ssh/id_rsa.pub',
                          '/home/username/.ssh/id_rsa', 'secret')) {
  echo "Public Key Authentication Successful\n";
} else {
  attackDetected("Public Key Authentication Failed", 100);
}
```

Code 26: Call attackDetected() whenever the authenticity of a connection fails

Path Traversal

User input is sometimes used to reference file names that will be read or written. A common tactic to avoid path traversal vulnerabilities[7] is to analyze the canonical path instead of the relative or the absolute path. However, watching for any difference between the absolute path and the canonical path may expose a path traversal attack:

[7] CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (https://cwe.mitre.org/data/definitions/22.html)

```
import java.io.File;
public class CheckPaths {
  public static void main(String [] args) {
    File f = new File(args[0]);
    try {
      String absoluteFile = f.getAbsolutePath();
      String canonicalFile = f.getCanonicalPath();
      if(!absoluteFile.equals(canonicalFile))
        System.out.println("Path traversal detected: "+f.getAbsolutePath()+" vs "+f.getCanonicalPath());
    }
    catch(Exception e) {
      System.out.println(e);
    }
  }
}
```

Code 27: Sample Java class for detecting path traversals

The previous code will detect the difference between `getAbsolutePath()` and `getCanonicalPath()` when analyzing user input. Three different user inputs will be analyzed to test the previous class: the first one is a correct value, the second value tries to exploit a path traversal, and the third tries to follow a link before accessing the file[8]:

```
$ ls -l /somefile /../somefile /tmp/somefile
ls: /../somefile: No such file or directory
ls: /somefile: No such file or directory
lrwxr-xr-x  1 user  wheel  11 Feb 29 08:24 /tmp/somefile -> /etc/passwd

$ java CheckPaths /somefile.txt

$ java CheckPaths /../somefile.txt
Path traversal detected: /../somefile.txt vs /somefile.txt

$ java CheckPaths /tmp/somefile
Path traversal detected: /tmp/somefile vs /etc/passwd
```

Figure 2: Sample output of CheckPaths

Anti-CSRF

Applications may include anti-CSRF[9] protection whenever they receive a request. The user may be a victim of an attack or may be an attacker testing for the vulnerability. The attack must be flagged with a medium score and/or eventually warn the user that they are the victim of an attack:

```
if(!verifyAntiXSRF(anti-xsrf-token))
  attackDetected("Anti-XSRF token invalid", 50);
```

Code 28: Call attackDetected() when detecting CSRF attacks

[8] CWE-59: Improper Link Resolution Before File Access ('Link Following') (https://cwe.mitre.org/data/definitions/59.html)
[9] CWE-352: Cross-Site Request Forgery (CSRF) (https://cwe.mitre.org/data/definitions/352.html)

Client Properties

Certain properties should be analyzed when connections are being established: the origin of the connection and when it is taking place.

Origin

The IP address is the first thing that will be checked before a user connects to the server. The application may restrict connections to certain IP addresses and/or it may use the IP address to identify the country from which the user is establishing the connection. Certain software companies like Google are implementing a defensive control here: whenever Gmail detects that you are logging in from an unknown country, it requires special user interaction before you are able to use their application.

```
if(isGeoLocationForbidden($session))
  attackDetected("Geo location is forbidden", 100);
```

Code 29: Call attackDetected() when a different or forbidden geo location has been used

Time of Access

This type of restriction has been in place since ancient times in UNIX systems. It is an unusual restriction, but certain applications may want to take actions whenever users are trying to log in outside of business hours.

```
if (date('H') < 8 || date('H') > 20)
  alertAdmin("The user logged in outside business hours");
```

Code 30: Call attackDetected() or alertAdmin() when a user logs in outside of business hours

Exceptions

Exceptions may be thrown in multiple places throughout an application when an attacker attempts to perform invalid operations. Payloads are never perfect on the first try, so it is important to flag these occurrences as attacks. Malicious requests should be logged to analyze the attack vectors and eventually stop processing the requests and/or the affected functionalities.

Consider the following list compiled by CERT[10] of Java exceptions that are areas of interest for attackers:

Exception Name	Description
java.io.FileNotFoundException	Underlying file system structure, user name enumeration
java.lang.OutOfMemoryError	Denial of service
java.lang.StackOverflowError	Denial of service

[10] Computer Emergency Response Team (CERT) is a trusted, authoritative organization dedicated to improving the security and resilience of computer systems and networks, which also define secure coding guidelines.

Exception Name	Description
java.net.BindException	Enumeration of open ports when untrusted client can choose server port
java.security.acl.NotOwnerException	Owner enumeration
java.sql.SQLException	Database structure, user name enumeration
java.util.ConcurrentModificationException	May provide information about thread-unsafe code
java.util.jar.JarException	Underlying file system structure
java.util.MissingResourceException	Resource enumeration
javax.naming.InsufficientResourcesException	Insufficient server resources (may aid denial of service)

Table 1: Java exceptions that may expose sensitive information[11]

Expected Exceptions

Applications catch different types of exceptions, some more dangerous than others depending on the context of the exception. The following compliant security policy implemented by CERT shows how to avoid exposing sensitive 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;
     }
   }
}
```

Code 31: Sample Java code from CERT to avoid exposing information with exceptions

[11] ERR01-J. Do not allow exceptions to expose sensitive information (https://www.securecoding.cert.org/confluence/display/java/ERR01-J.+Do+not+allow+exceptions+to+expose+sensitive+information)

The catch condition in line 11 should never occur, as it could be a symptom of a path traversal attack. It is a perfect place to trigger a warning that an attack has been detected:

```
...
11:     } catch (IOException x) {
12:       System.out.println("Invalid file");
13:       attackDetected("Path traversal detected", 50);
14:       return;
```

Code 32: Call attackDetected() if someone is trying to exploit a path traversal

Uncaught Exceptions

Attackers can use information from uncaught exceptions to focus their attacks more precisely. The impact of this issue depends on the type of exception being triggered. The errors may assist an attacker to exploit an issue not previously considered by the developer.

This is the output of the previous class when executed without parameters:

```
$ java ExceptionExample
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
   at ExceptionExample.main(ExceptionExample.java:9)
```

Figure 3: Output of the ExceptionExample class

Line 7 references `args[0]` without verifying if it has the required arguments[12], so the first thing it does is to throw a `java.lang.ArrayIndexOutOfBoundsException`. This exception is not properly handled, and according to CERT, all exceptions should be handled appropriately[13]. CERT also states that exceptions must be prevented while logging data, but the `System.out.println()` calls (lines 9, 13, and 20) are in violation of secure behavior[14].

Finally, the purpose of the previous class is to conceal "*any information about files outside c:\homepath*" (line 8). That statement is false, since files in the directory "`c:\homepathfake`" are outside of "`c:\homepath`", and they would be considered as correct. Line 8 should reference the directory with a backslash at the end "`c:\\homepath\\`" instead of using "`c:\\homepath`".

Even trusted, security-compliant code may be subject to attacks, and that is why all uncaught exceptions should be analyzed along with the requests that produce them. A global catch statement should be defined for uncaught exceptions thrown by the application.

---

[12] MET00-J. Validate method arguments (https://www.securecoding.cert.org/confluence/display/java/MET00-J.+Validate+method+arguments)

[13] ERR51-CPP. Handle all exceptions (https://www.securecoding.cert.org/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions)

[14] ERR02-J. Prevent exceptions while logging data (https://www.securecoding.cert.org/confluence/display/java/ERR02-J.+Prevent+exceptions+while+logging+data)

```
public class uncaught {
  public static void main(String[] args) throws Exception {
    Thread.currentThread().setUncaughtExceptionHandler(new
  Thread.UncaughtExceptionHandler() {
      public void uncaughtException(Thread t, Throwable e) {
        logger.error("Uncaught exception", e);
        attackDetected("Uncaught exception", 100);
        System.exit(1);
      }
    });
    throw new IllegalArgumentException("uncaught exception for testing");
  }
}
```

Code 33: Example Java code to detect uncaught exceptions

To provide a basic virtual patching[15] on the fly, you may want to prevent the functionality[16] from being called again with the same parameters until it is reviewed.

Patterns

Lack of anti-automation controls allows attackers to obtain information by issuing unlimited requests. Patterns can be identified when analyzing these requests.

Password Looping

A basic protection against brute-force attacks involves restricting the number of passwords that can be attempted for each account. However, a common pattern attackers use to avoid that protection is to attempt to log in to each user account only a few times and then move on to the next account before the first one is blocked. An attacker may loop through several different user accounts until they are able to break in without blocking them.

The defensive control should not only be put on the account being targeted, but also on the IP address making the connections[17]:

```
if (!login($user, $pass)) // sample failed login functionality
  attackDetected("Password attempt", 10);
```

Code 34: Call attackDetected() whenever a user is unable to login

Frequency

The number of requests initiated by manual browsing does not compare to the requests per minute performed by attack tools. Attack tools and certain attacks (such as padding attacks, entropy analysis of cookies, measuring the timing of web servers, etc.) try to obtain the largest amount of information in the least amount of time.

[15] Virtual Patching Best Practices (https://www.owasp.org/index.php/Virtual_Patching_Best_Practices)
[16] CWE-390: Detection of Error Condition Without Action (https://cwe.mitre.org/data/definitions/390.html)
[17] CWE-307: Improper Restriction of Excessive Authentication Attempts (https://cwe.mitre.org/data/definitions/307.html)

The number of requests per minute may be also analyzed by the application by saving the current time (lines 2 and 4), then adding a hit per request (line 5). In case the number of requests in the last minute is higher than 100 (line 6), consider the action to be an attack (line 5):

```
1: if($userSession ["requests_last_minute"] < (time()-60) ) {
2:   $userSession["requests_last_minute"] = time();
3:   $userSession["amount_requests_last_minute"] = 0;
4: }
5: $userSession ["amount_requests_last_minute"] += 1;
6: if($_SESSION["amount_requests_last_minute"] > 100) { // Sample max value
7:   attackDetected("Too many requests per minute", 100);
8: }
```

Code 35: Pseudocode from the defense class to measure requests per minute

The previous control has also been implemented by Fail2Ban. It analyzes the log files of the web server and bans the ones that show signs of malicious actions[18].

Post-execution Controls

If an attacker is able to circumvent the pre-execution and embedded execution controls, the last line of defense before they get a response will be the post-execution controls. The response may contain valuable information that could aid an attacker.

Sensitive Information

Passwords are one of the ultimate goals for attackers, whether they are in the operating system (for example, in /etc/passwd or /etc/shadow) or the database (for example, the user's table containing the passwords hashes). Fake user accounts and/or hashes could be introduced in these places and used to trigger exfiltration attack detection whenever these special strings are found in responses:

```
if(strstr($response, "fakeAdminUser")) // fake admin user in /etc/passwd
  attackDetected("Passwords leaked", 100);
```

Code 36: Call attackDetected() when a password exfiltration is detected

Attackers may not be able to retrieve the entire file or database table containing passwords all within one single request, but they may retrieve passwords line by line. This means that the locations of the fake passwords should be properly analyzed in advance.

Operating system files and environmental information is also an important piece of information that can be detected before it is retrieved. If an attacker is able to list operating system files, certain filenames could be used to trigger an alarm whenever they are found in responses. Additionally, special strings could be put in the web server’s environment in case the attacker is traversing the `/proc` directory to look for information in the environ files (valuable information when they do not have a path disclosure). If the

[18] Fail2Ban (http://www.fail2ban.org/)

"`secrethiddendirectory`" is found within the response, it should be considered a critical-risk attack:

```
if(strstr($response, "secrethiddendirectory"))  // fake file/directory
  attackDetected ("Files leaked", 100);
```

Code 37: Call attackDetected() when a filename exfiltration is detected

Whenever an attacker is able to exfiltrate sensitive data from your system (passwords or files), a critical action should take place. This means disconnecting the application from the network to perform forensic analysis after the attack has been detected. Any further interaction may only benefit the current capabilities of the attacker.

CERT states that programs must operate only on files in secure directories[19]. As shown in Figure 4, if the path contains any symbolic links, the routine provided by CERT will recursively invoke itself (in green) on the linked-to directory and ensure it is also secure. A symlinked directory may be secure if both its source and linked-to directory are secure (in blue). An extract of the complete method is shown below:

```
// If any parent dirs (from root on down) are not secure,
// dir is not secure
for (int i = 1; i <= file.getNameCount(); i++) {
  Path partialPath = Paths.get(file.getRoot().toString(),
                               file.subpath(0, i).toString());

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

Figure 4: Extract code from CERT for the method isInSecureDir()

The method `isInSecureDir()` does not analyze the possibility of hard links. This property can be retrieved in Java when requesting the integer property value `unix:nlink`. This means that if an attacker sets a hard link to a file in a shared directory, all of the file's contents will also be written to the hard-linked file in the shared directory. This method may be used to retrieve sensitive information from the system.

Timing Information

The amount of time required by a function when executing a request[20] can provide valuable information to an attacker. Authentication is a common target for attackers when

[19] FIO00-J. Do not operate on files in shared directories (https://www.securecoding.cert.org/confluence/display/java/FIO00-J.+Do+not+operate+on+files+in+shared+directories)

[20] CWE-208: Information Exposure Through Timing Discrepancy (https://cwe.mitre.org/data/definitions/208.html)

performing a blind attack against the characters of a password[21]. Introducing a small random delay of microseconds in the response will reduce the probability of this type of attack succeeding and will go unnoticed by regular users:

```
function login($user, $pass) {
  ...
  usleep(mt_rand(0,1000)); // delay 0 to 1000 microseconds
 return $loginStatus;
}
```

Code 38: Add a small delay in login functions to avoid disclosing execution time

Moreover, requests taking more time than expected could also act as a warning for potential attacks. If attackers are able to cause requests to be delayed (by introducing a delay on purpose with a blind SQL injection[22] or an entity expansion[23]) it may indicate that code execution, SQL injection, or denial-of-service attacks are possible. To properly control for this, application responses should be measured to avoid false positives when responses take longer than expected.

The maximum amount of wait time should be as short as possible without interfering with the delay of functionalities:

```
if( ($start_time+5) < time()) // if it takes more than 5 seconds..
  attackDetected("Too much time", 20);
```

Code 39: Call attackDetected() when processing the function takes more than certain amount of time

Size Information

In addition to looking for the most time-consuming functionalities, attackers also look for the most bandwidth-consuming functionalities, with the goal of depleting bandwidth resources.

Additionally, attack tools often determine if their actions had an impact based in the size and content of the application responses. This can easily be disrupted by adding a line of code that introduces a small random number of whitespaces or comments to the response to mess with the attacker’s analysis. This can be implemented in multiple random places in the application, particularly in headers and footers since they are commonly implemented by all of the functionalities:

[21] Web Timing Attacks Made Practical (https://www.blackhat.com/docs/us-15/materials/us-15-Morgan-Web-Timing-Attacks-Made-Practical-wp.pdf)
[22] Blind SQL Injection - Time Based (https://www.owasp.org/index.php/Blind_SQL_Injection)
[23] XML Entity Expansion (http://projects.webappsec.org/w/page/13247002/XML%20Entity%20Expansion)

```
1: echo str_pad(' ', mt_rand(0,10)); // add 0 to 10 random whitespaces
```

Code 40: Add random spaces to always have different sizes in responses

Environment Information

When it comes to web applications and web servers, there are different pieces of environment information that provide knowledge to attackers. You do not want to share all of the available information and how their actions are affecting you. For example, even though the `Date` header must be sent according to the HTTP standard[24], it provides information to attackers who want to predict the values of pseudo-random number generators (PRNG)[25]. In the same way that you should avoid disclosing that an "`HTTP 500 Internal Server Error`" has occurred, you may want to avoid disclosing internal information about the web server that can be used to perform other attacks.

As was previously stated, HTTP errors disclose how an attacker's actions affect the application and should be hidden. Keep the information to yourself, give the attacker a fake response[26], and then work on solving the problem that caused the original request. This could be either set up in the web server or in the application:

```
1: header("HTTP/1.1 200 OK"); // force an OK response
```

Code 41: Send a fake HTTP OK response whenever an error is triggered

Attack Yourself

The defensive functionalities should be tested to eliminate false positives, verify that the information is being shared across the required platforms, and confirm that alarms are working as expected. A penetration test can provide you with this information and help you identify any adjustments that need to be made.

[24] Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content (https://tools.ietf.org/html/rfc7231#section-7.1.1.2)

[25] PRNG: Pwning Random Number Generators (https://media.blackhat.com/bh-us-12/Briefings/Argyros/BH_US_12_Argyros_PRNG_WP.pdf)

[26] Defense by Numb3r5 - Making problems for script k1dd13s and scanner monkeys (https://www.defcon.org/images/defcon-21/dc-21-presentations/Riley/DEFCON-21-Riley-Defense-by-Numbers-Updated.pdf)

During an Attack

If you know you are under attack, what should you do? There are three basic steps you want to take:

1. Log and share information about the attack
2. Inform responsible parties about the attack
3. Take appropriate action

Log the Attack

Once applications go live, they should log all of the information and share it internally.

Logging should provide non-repudiation, timestamps, and information about the attacker's actions. It is important to save the information that will allow a post-mortem analysis to correlate which functionalities the same attacker executed.

The following pieces of information should be saved:

- Timestamp
- Affected application
- Attacker's IP address
- Attacker's username (if they were logged in or tried to log in)
- Attacker's real cookie
- Affected filename
- Type of attack
- Score of the attack

This information can be used to track attackers and take actions whenever they reach the maximum score allowed. Furthermore, whenever certain critical conditions are triggered (for example, "Expected Exceptions") the filename could be quarantined until further analysis is performed. These pieces of information may ultimately be used in legal actions against the attacker.

Information regarding whom is considered an attacker must be available for other hosts and applications within your network before even informing about the attack. All systems in your network should verify the attacker's database to avoid dealing with malicious actors. Attacks may focus first on the systems that require the least amount of effort to compromise, and then may try to escalate to target your main application from within your network.

It is possible to get lists of malicious IP address from third parties. This information can be provided for free[27] or by paid subscriptions[28]. However, these third parties do not provide clear information on how the information was detected, what was detected, when it was detected, and who detected which activity.

Inform about the Attack

Once an attack has been detected, an alarm should be triggered to inform the responsible person(s). First the information about the attack and the attacker should be gathered before alerting the administrator:

```
1: $alert_info = "The last attack from the user was: ".$attack;
2: if($attackScore >= $LIMIT)
3:   $alert_info = $alert_info.". The user reach the maximum score";
4: else
5:   $alert_info = $alert_info.". The user reach the maximum score because of a
   series of events";
6: $alert_info = $alert_info.".\nAttacker details:\n";
7: $alert_info = $alert_info."IP: ".$session_parameters['ip']."\n";
8: $alert_info = $alert_info."User: ".$session_parameters['user']."\n";
9: $alert_info = $alert_info."Cookie: ".$session_parameters['cookie']."\n";
10: $this->alertAdmin($alert_info);
```

Code 42: Pseudocode from the defense class with the attack/attacker information

Lines 2-5 will inform how the user reached the maximum score. Lines 6-9 will provide some information about the attacker and then the alert will be triggered. The alert could be an SNMP message, a visual pop-up, or an email message. The following code sends an email message once the score exceeds a certain threshold:

```
function alertAdmin($alert_info) {
  mail("admin@yourdomain.org", "attacker detected", $alert_info);
}
```

Code 43: Pseudocode from the defense class to alert the admin about the attack information

The first implementation of this framework was deployed for an application on an internal network. After sitting quietly for six months, the first real email of an attack was sent. A vulnerability scanner was detected, as they were in the recognition phase. After receiving the alarm, the issue was escalated to the IT team, and the attacker was quickly identified.

[27] IP Address Blacklist Checker Tool (http://www.ipvoid.com)
[28] McAfee GTI Reputation & Categorization Services (http://www.mcafee.com/us/threat-center/technology/gti-reputation-technologies.aspx)

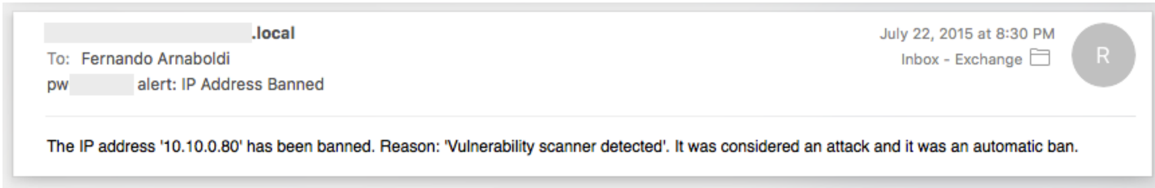

Figure 5: First real detection of an attack in an internal network

Take Action

Magruder's principle states that it is generally easier to induce an enemy to maintain a pre-existing belief than to present notional evidence to change that belief. Thus, it may be better to examine how an enemy's existing beliefs can be turned to your advantage rather than attempting to change their beliefs.[29]

Without revealing your methods, and depending on the scenario, you may want to take actions to wear down the attacker. It is important that these approaches are low cost and easy to maintain when dealing with long-term attacks. Doing a mixture of different techniques may push an attacker to spend more time and computational power in worthless areas. This may give you time to tighten up your application and better deal with their actions.

Distract

You can induce an attacker to penetrate fake areas of your application. Attackers may want to believe that certain functionalities will be present in your application, such as administrative functionalities. Attackers may try to access the "`/admin`" URL on your website, and you could prepare a fake authentication system to keep them entertained.

Another possibility would be to show them incorrect information to increase the difficulty to distinguish true vulnerabilities from false vulnerabilities. A common approach is to change the 404 responses to show always 200 OK responses, and try to disguise when files are not found. Another possibility would be to introduce "`HTTP 500 Internal Server Error`" error codes to force the attacker to analyze properly working requests or eventually cause fake exception codes to be triggered.

Slow Down

In certain scenarios, it is worth slowing down the attackers. When they are sending multiple requests per minute, you may want to include an option to slow down the attacks. Faking a higher response time is beneficial from two points of view:

- They may be real users using the same IP address, and slowing them may keep your application up and running.

[29] Battlefield Deception - Magruder's principle (http://fas.org/irp/doddir/army/fm90-2/90-2ch1.htm)

© IOActive, Inc. All Rights Reserved.

- If they are real attackers, faking incapacity may cause them to believe that they are achieving their objective.

Ultimately, you may decide to stop responding completely to make the attacker believe that they have accomplished their goal (if it was a denial of service) or that they will be unable to do so (if the attack requires numerous requests in a specific time frame).

Stop

Blocking is the most common defense mechanism in systems and applications. A common scenario happens today when an attacker tries multiple incorrect passwords on a user's account. The targeted account is locked to prevent an attacker from getting into it. Even though this action protects the victim from being hacked, it does not do anything to address the attacker.

Once an attacker reaches a certain score, the attacker could be banned from the application. The ban can be set up to last for a specific amount of time or they could require an administrator to unban them. These scenarios should take into consideration what the consequences would be if the attacker had multiple IP addresses (for example, if they are using IPv6) or if the attacker is using an IP address shared with other legitimate users. Depending on the context, it may be preferable to block entire subnets and then analyze the best course of action. The ultimate blocking action is to disconnect the affected system from your network to avoid any further interaction and perform computer forensic analysis.

Bait

Developers may include snippets of code in applications that whenever they are hacked, enable them to prosecute attackers. These bait functionalities or pieces of information must not have a negative impact on the image of the company. Some possibilities are:

- A small portion of the application can be created with sequential values for usernames in the URL. Whenever someone tries a sequential value they could be illegally accessing other user's account confidential information, which may be punished.

- An authentication with a warning message disclosing that it is illegal to connect to it without authorization. This authentication may use default usernames and passwords in order to be able to rightfully accuse attackers of breaking into restricted locations.

- Give information on a real credit card and/or bank information after the attacker is detected and take actions whenever they are used.

Some of these actions could already be in place to lure attackers and prosecute them as criminals for breaching applications.

Conclusions

Security should be included in the design of applications, pairing developers with hackers. Implementing a defensive layer allows applications to control what is happening internally and control different situations.

An arms race can take place between attackers and defenders when implementing a defensive layer. Attackers may use more resources and start changing their methods to attack applications. When this happens, the defense should be improved and updated to cope as attack techniques evolve.

Defensive controls can also be used to expose potential flaws in the application. Attackers may trigger flaws using novel mechanisms not previously considered, which can be further analyzed when they are exploited.

Appendix A: Score Table

Every attack must have a probability associated with the corresponding likelihood. This probability can act as the score for a potential attack. The framework considers that whenever the score reaches 100, the user should be considered an attacker.

Whenever a control requires only the "IP or session" to be verified, this means that if a single entry point is defined for the application, it is only required to have the control one time. Mostly this will apply to pre-execution and post-execution controls.

The following scores may serve as a guide and should be adjusted per application:

ID	Category	Type of Attack	Probability of Attack	Requires
1	Pre execution	Malicious HTTP Method	25	IP or session
2	Pre execution	Malicious URL	10	IP or session
3	Pre execution	Invalid HTTP Protocol Version	100	IP or session
4	Pre execution	Incorrect Hostname	100	IP or session
5	Pre execution	Forced Browsing: Invalid URI	20	IP or session
6	Pre execution	Forced Browsing: Non-existent Resource	5	IP or session
7	Pre execution	Forced Browsing: Backup File	100	IP or session
8	Pre execution	Forced Browsing: Existent Resource Anonymous	20	Auth user
9	Pre execution	Forced Browsing: Privileged Resource Authenticated	100	Auth user
10	Pre execution	User-Agent: Malicious String	100	IP or session
11	Pre execution	User-Agent: Did it change?	100	Session
12	Pre execution	Cookies: Concurrent Session	25	Session
13	Pre execution	Trap: Fake robots.txt Entry	100	IP or session
14	Pre execution	Trap: Fake Hidden URL	100	IP or session
15	Pre execution	Trap: Fake Cookie	100	Session
16	Pre execution	Trap: Fake Input Field	100	Session
17	Execution	Check: Incorrect HTTP Method	100	IP or session
18	Execution	Check: Missing Parameter	20	IP or session
19	Execution	Check: Extra Parameters	20	IP or session
20	Execution	Check: Unexpected Values	100	IP or session
21	Execution	Check: MiTM	100	IP or session
22	Execution	Check: Path Traversal	100	IP or session
23	Execution	Check: Anti-CSRF	50	Session

ID	Category	Type of Attack	Probability of Attack	Requires
24	Execution	Client Properties: Origin	100	Auth user
25	Execution	Client Properties: Time of Access	0	Auth user
26	Execution	Exceptions: Expected	50	IP or session
27	Execution	Exceptions: Uncaught	100	IP or session
28	Execution	Patterns: Password Looping	10	IP or session
29	Execution	Patterns: Frequency	100	IP or session
30	Post execution	Sensitive Information	100	IP or session
31	Post execution	Time Information	20	IP or session

Table 2: Defense score table

Appendix B: Tool Identification

Tool	User-Agent	IP-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 XSS Exploit Framework	no	no	no	no	no	yes	no
Zed Attack Proxy	no	yes	no	no	no	yes	no

Table 3: Tool identification

© IOActive, Inc. All Rights Reserved.

About Fernando Arnaboldi
Fernando Arnaboldi is a senior security consultant at IOActive specializing in penetration testing and code reviews on multiple platforms. He has 20 years of development experience in a variety of programming languages and has presented in the past in security conferences such as Black Hat, DEF CON and OWASP AppSec USA.

About IOActive
IOActive is a comprehensive, high-end information security services firm with a long and established pedigree in delivering elite security services to its customers. Our world-renowned consulting and research teams deliver a portfolio of specialist security services ranging from penetration testing and application code assessment through to semiconductor reverse engineering. Global 500 companies across every industry continue to trust IOActive with their most critical and sensitive security issues. Founded in 1998, IOActive is headquartered in Seattle, USA, with global operations through the Americas, EMEA and Asia Pac regions. Visit www.ioactive.com for more information. Read the IOActive Labs Research Blog: http://blog.ioactive.com. Follow IOActive on Twitter: http://twitter.com/ioactive.