HOW TO REVEAL APPLICATION VULNERABILITIES? SAST, DAST, IAST AND OTHERS

PT Application Inspector Team

Security analysis of applications is commonly divided into two areas: static analysis and dynamic analysis. They are often incorrectly identified with white-box and black-box testing, respectively.

Contrary to common belief, dynamic analysis can be applied when an application and its source code are available. Moreover, dynamic analysis sometimes proves to be much more efficient here than static analysis. Another popular misconception is that source code analysis is equal to static analysis, while static analysis is applicable to compiled applications, too. Needless to say that today, various JIT solutions like .NET MSIL and Java Bytecode are widely used, which blurs out the difference between analyzing source code and compiled applications.

To further complicate matters, often times marketing language gets introduced that over simplifies the technical analysis methods it attempts to describe. For example, some Analysts interchangeably use: Interactive Application Security Testing (IAST), which in fact refers to dynamic analysis, Dynamic Application Security Testing (DAST) and Static Application Security Testing (SAST). However, to clear up any confusion, let me define some long-standing terms.

- DAST — dynamic (i.e. requiring execution) application security analysis without access to the server-end source code and runtime environment.
- SAST — static (i.e. not requiring execution) application security analysis with access to the server-end and client-end source code (along with its derivatives) and runtime environments.
- IAST — dynamic application security analysis with access to the server-end source code and runtime environment.
- Source code analysis — static or dynamic analysis with access to the server-end and client-end source code (and its derivations) and runtime environments.

In other words, DAST represents black-box dynamic analysis (at least, for server end), SAST represents white-box static analysis, and IAST represents white-box dynamic analysis.

DAST: The Good, the Bad and the Ugly

Black-box dynamic application analysis is the most simple and widespread method of vulnerability testing. In fact, every time when we insert a quote into a URL or enter '>, it is this complex test. It is actually fault injection into an application (aka fuzzing) implemented by emulating the client end and sending well-known invalid data to the server end.

The simplicity of this method results in a large number of implementations; Gartner's Magic Quadrant is full with competitors. Moreover, DAST engines are included in most compliance and vulnerability management systems such as MaxPatrol (Symantec Control Compliance Suite is perhaps the only exception). Noncommercial solutions are also widely used. For example, there is a basic (very basic) DAST module in Nessus and more advanced mechanisms in w3af and sqlmap.

The DAST advantages are its simplicity and the fact that there is no need to access the application server. Another good feature is its relative independence of the application platform, framework and programming language. You can use this additional information about applications to improve the analysis performance, but as an optional optimization.

However, DAST has its flipside.

- Black-box method fails to reveal many attack vectors.
- For more complex clients and protocols, analysis efficiency dramatically decreases. Web 2.0, JSON, Flash, HTML 5.0, and JavaScript applications require either dynamic (i.e. emulation of JavaScript execution) or static (Flash grep or taint analysis of JavaScript) client-end examination. It considerably complicates the fuzzer client turning it into nearly a fully featured browser.
- Nonzero probability of integrity and availability violation (e.g. if structures like 1=1 will get into UPDATE through SQL Injection).
- Long execution time. Your humble narrator has hardly ever seen a DAST utility finish its analysis of a rather wide-branching site — the testing window always closed first. The Pareto principle suites the situation rather good — 80% of vulnerabilities come from 20% of time spent.
- It is difficult to find many vulnerability types. For example, cryptographic errors like weak generating methods for cookie or session ID (except the simplest cases) are poorly detected by DAST.

SAST: Quick and Dirty

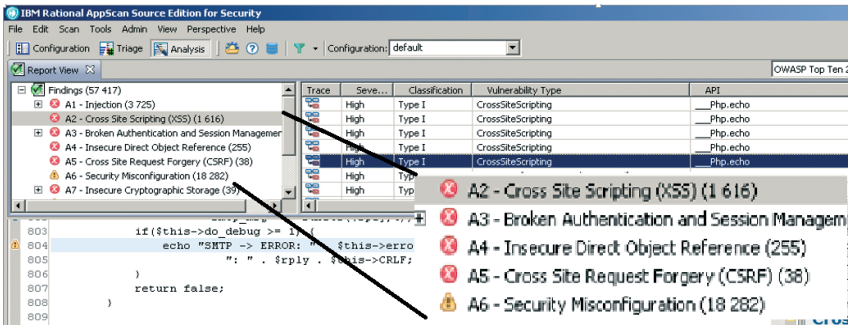
Now let's talk about SAST disadvantages. First, there is no integrated engineering or scientific approach (because of objective difficulties). As a result, every developer invents his/her own way, and then marketing guys wrap it into glossy pseudo-scientific package.

Secondly, most static analysis methods generate a high number of potential vulnerabilities that turn out to be false positive at great expense to resources.

Thirdly, SAST is unable to detect certain vulnerability types (bit.ly/1nobul5). Finally, a static approach implies restrictions. There is a number of examples: code generated on the fly, storing code and data in DBMSs, file systems etc. Extra effort depends on code languages (and even versions) and frameworks. Just a language is not enough to detect entry and exit points and filtering functions, a developer needs to find libraries and frameworks used in real life.

```
<?php

$yii = dirname(__FILE__).'/framework/yii.php';
$config = dirname(__FILE__).'/protected/config/bil.config.php';
$client_config = '/usr/local/bil/client.main.php';

require_once($yii);

if (file_exists($client_config) && is_readable($client_config)){
    $config = CMap::mergeArray(
        require_once($config),
        require_once($client_config)
    );
}

Yii::createWebApplication($config)->run();
```

Dynamic inclusions together with transparent application initialization



Just 1600+ XSSs and 18 000+ configuration flaws! Efficiency of analysis is perfectly shown, isn't it?

POSITIVE TECHNOLOGIES

Cross a Hedgehog with a Snake

SAST and DAST have advantages and disadvantages; therefore the community proposes an idea to unite these approaches or to use a hybrid method that takes only the good from each approach. While this idea in concept is not hard to imagine, it has proven to be very hard to execute successfully. In fact, in the past several years, there have been at least 3 separate attempts to solve this problem.

The first attempt included correlating and cross-utilizing DAST and SAST results. This approach is intended to improve the coverage of dynamic analysis by applying the results of the static one and to reduce the number of false positives. For example, IBM chose this path.

However, it became quickly evident that the advantages of this approach were overestimated. Additional entry points transferred from SAST to DAST without context (or, in our terms, without additional requirements) often only increased the processing time. Think of SAST detecting 10,000 combinations of URLs and parameters and sending them to DAST; at that, 9,990 of them require authorization. If SAST doesn't "explain" to DAST that authorization is required and doesn't provide information on how to authorize, then the scanner will be senselessly knocking at all these URLs and operating time will increase 1,000 times, almost without any result change.

The main problem, however, was actually incompatibility of DAST input and SAST output. In most cases, static analyzer output looks as follows: "insufficient filtration in line 36, XSS is possible." However, the DAST native format is an HTTP request like */path/api?parameter=[XSS]&topic=important with* vulnerability type specified and preferably with a set of values for fuzzer considering filtration functions.

Pay regard for an important parameter topic=important. It is this requirement to be fulfilled for fuzzer to get into the necessary vulnerable API. Nobody guarantees that the parameter will be vulnerable when addressing */path/api?parameter=[XSS]&topic=other.* How on earth will the static analyzer know it? No idea…

Various additional modules like mod_rewrite, frameworks, web server settings, etc. make the problem even more complex.

In short, it didn't work.

Hybrid Theory

Another approach considered was IAST (Interactive or even Intrinsic Application Security Testing). IAST is in fact an extension of dynamic analysis including access to the server-end source code and runtime environment (in the course of fuzzing using DAST). For this purpose, either instrumentation of web server, framework or source code is used or built-into tracing tools.

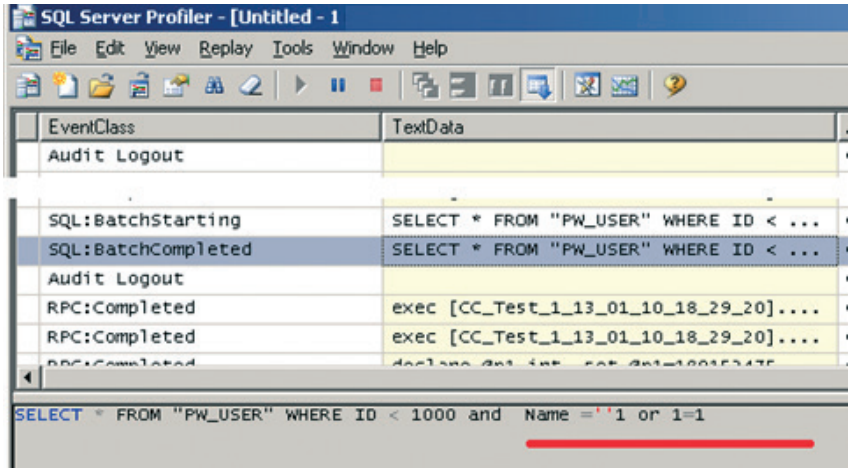
For example, you can effectively use the results obtained by SQL Server Profiler or similar utilities to detect SQL Injection. These utilities show what passed through all filtering functions and actually reached the server.

When IAST dynamic taint analysis was presented, once again, to the AppSec community, it captured a large number of supporters who praised its advantages. For example, it allows one to improve the dynamic analysis efficiency by tracking distribution of fuzzer requests through different levels of an application (it helps reduce false positives and detect Double Blind SQL injection). Moreover, instrumentation on different levels of an application allows one to detect delayed attacks such as Stored XSS and Second Order SQL Injection, which are hardly recognized by either SAST or DAST.

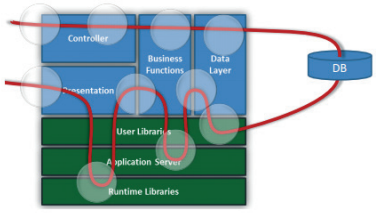
It is also important that this approach provides a solution to the problem of URL-to-source mappings, i.e. mapping of application input points and source-code lines. The three-stage solution, depicted below, is complex and requires instrumentation code for the server, but it does provide some result.

IAST Disadvantages

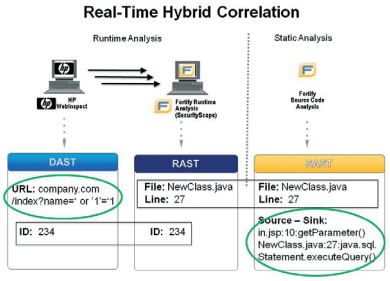
However, IAST has its disadvantages. First, it is necessary to perform code instrumentation and/or install an agent for dynamic instrumentation of web servers, DBMSs and frameworks. Obviously, it is not (always) applicable to (all) production systems. Furthermore, compatibility and performance issues arise.



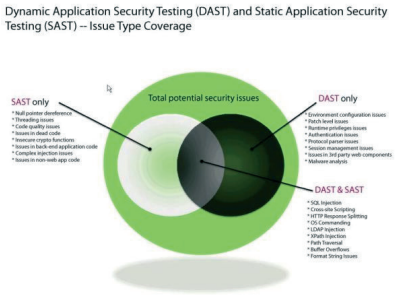
It seems to me I made a little IAST again…



Stored XSS and its SpyFilter tracing



Hybrid analysis, HP vision (RAST=IAST)



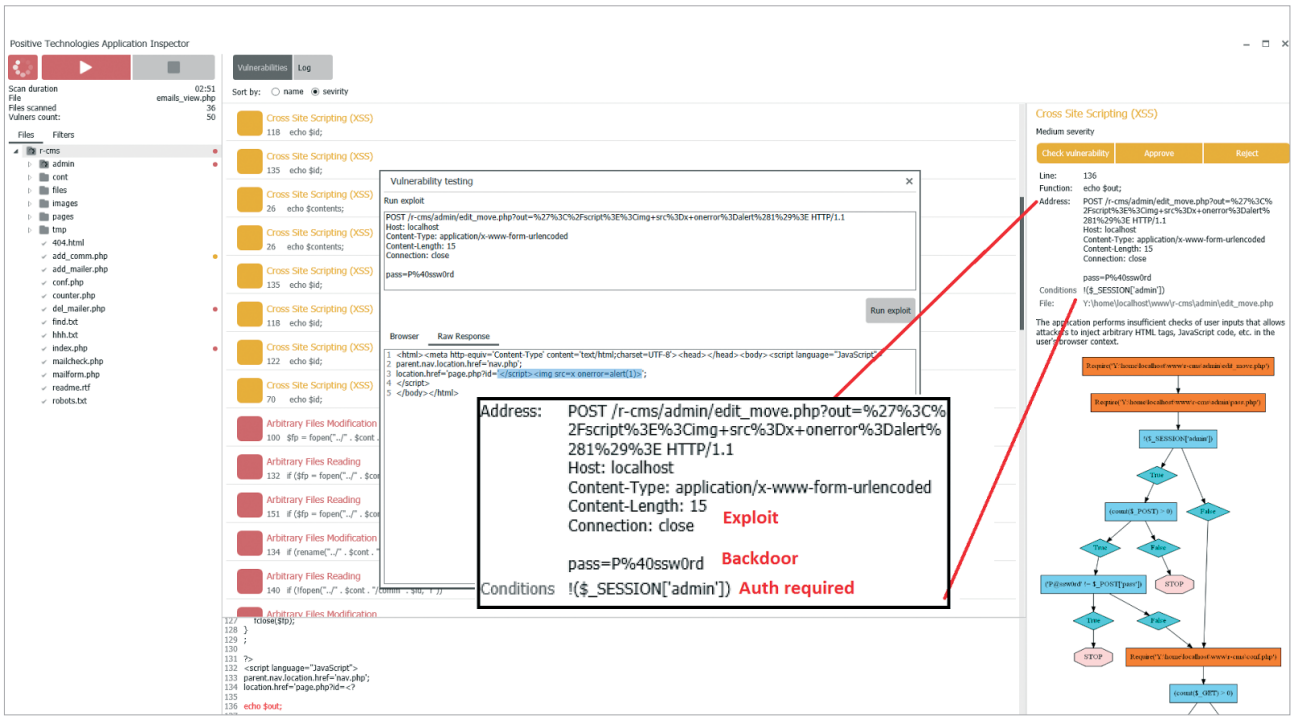
It works like this (bit.ly/Q9rZwl)

Positive Technologies Studied Oracle Siebel Security

In studying Oracle Siebel CRM security, the experts at Positive Technologies discovered multiple security flaws that could trigger remote command execution, internal network resources and file system availability, denial of service and sensitive data disclosure. Moreover, Oracle Database was found to contain a vulnerability, which allows a malware user to access remote resource contents and conduct DoS attacks. Vyacheslav Egoshin, Nikita Mikhalevsky, Alexey Osipov, Dmitry Serebryannikov, Dmitry Sklyarov, Alexander Tlyapov, Sergey Shcherbel, and Timur Yunusov took part in the study.

A Critical Patch Update (CPU) fixing the above mentioned flaw was released in mid-October 2013.

POSITIVE TECHNOLOGIES

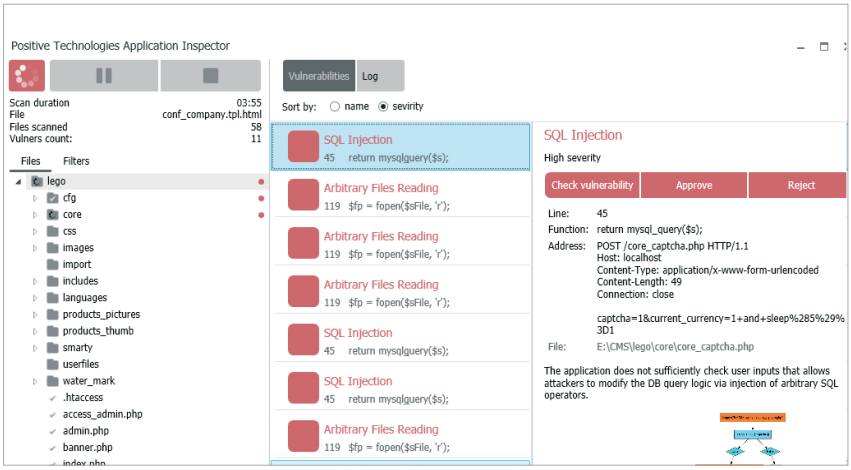


Exploit, backdoor and ~~two smoking barrels~~ additional requirement (bit.ly/1p6ER6c)

Additionally, IAST currently represents an extension of DAST (by the way, Gartner directly denotes it in their blog, gtnr.it/1iTvUpy) and inherits both positive and negative features of this method. It is whispered that the pure IAST (bit.ly/1eKmNVJ) is coming, but it is going to be more like an Application IDS/Firewall that can incidentally reveal vulnerabilities.

Returning to the subject, IAST serves as an intermediate mechanism and allows us to solve the problem of SAST and DAST compatibility, but only partly. In some cases, especially having a worthwhile SAST, it can give quite a good result.

You can find some more criticism of the hybrid analysis in the article A Dose of Reality on Hybrid Analysis by Chris Eng (bit.ly/1qDNNxd). Let me note that most of his arguments are applicable both to the simple correlation of SAST and DAST results and hybrid analysis using IAST.



Double Blind SQLi here, a Time-Based one needed...

Call to Arms

It is clearly seen that IAST represents an approach that is too complex. There is a strong need for a more efficient solution. This new method has no buzzword yet, but its aspects are described in scientific papers more and more often. The essence of this approach is to combine static and dynamic analysis without any intermediate link. The basic principle is the same as in IAST, but the static analysis (which is potentially more comprehensive) is considered here to be the principle one. To solve the problem, SAST should prepare data for verification in DAST-suitable format, e.g. in the form of an HTTP request with additional requirements and a set of parameter values for fuzzing.

How to solve the format problem? It is a theme for a separate article. However, let me note that there is nothing impossible. By the way, the new approach also allows us to integrate SAST with Web Application Firewall, which will help eliminate found vulnerabilities more quickly.

One more thing. Please, do not take this the wrong way. I am not against IAST, but I'm for it. Some acridity is a respond to the statements like "IAST>SAST+DAST!" This method has its niche. Furthermore, it helps implement the concept of continuous monitoring, which is trendy today. However, you can obtain the same results without fuzzing the entire application. In some cases, you do not even need to run it… Let's talk about it some other time.

Positive Technologies' Research Helped World's Largest Companies

Timur Yunusov and Alexey Osipov unveiled their research "XML Out-of-Band Data Retrieval" at Black Hat Europe in spring 2013. From this research, an automated technique called XXE OOB was created and used to help detect and fix new types of vulnerabilities in Microsoft Office, Invensys Wonderware, Oracle software, and Siemens SCADA components. Such security flaws were even detected in ModSecurity, a popular open source firewall designed to protect web applications. By the end of the year, their research was included in the Top 10 Web Hacking Techniques, an annual list of the most significant vulnerability analysis techniques. On another note, a method described in the article "Random Numbers. Take Two" by Dmitry Nagibin, Arseny Reutov, and Timur Yunusov made the Top 10 list in 2012, placing fourth.

© 2014 Positive Technologies.