LEVVEL

DevOps for Enterprise Security

How we helped lock down one of the largest banks in the world from threats

2018

As developers we leverage DevOps tools to streamline verification tasks with a goal to verify applications work correctly, but security testing is very different from testing feature functionality. Security testing identifies risks and vulnerabilities in the kernel, operating system, applications, stack, network, and infrastructure.

In this report, we will introduce how we help lock down web applications and REST API service layers using free open source tools (FOSS) with well-adopted DevOps toolchains, and if you have the budget there are many commercial solutions available too. While some of these tools can be applied to non-web technologies, things like kernel hardening or malware integration testing are just too complex to tackle at the same time as this introductory report. Stay tuned for more or reach out if you want to talk through how we approach automating those use cases.

For Information Security, 2018 started with a bang, showcasing Intel's Spectre and Meltdown exploits. The fact that all intel chips allow Javascript to access memory in the kernel space is frankly incredible.

Now is the time to start putting processes with the right tools in place to help mitigate security issues to your business. This document shares how Levvel is helping some of the largest financial institutions (and companies in all industries of all sizes) address their security concerns. The past few months have been eye-opening with all the hacks and exploits affecting companies large and small. This report will highlight how we helped create a secure compute environment for a large financial institution's application stack written in python and extensible to most modern programming languages.

Where do we begin, and how much time and money is required?

Before we dive in we wanted to keep this report shorter, which meant we will not be determining your organization's overarching security strategy. Instead we will share how

we are effective in helping our clients with FOSS tools to reduce their business costs to deliver features to their customers. In the application development space, the adoption of DevOps toolchains is essential to keeping up with the competition. We have seen that with the right solution, security teams can reduce their costs as well.

The scope of this report focuses specifically on how to lock down your applications with tried-and-tested DevOps tools integrated with the latest enterprise-grade security analysis toolchains. These tools are helping organizations automate testing while reducing the cost to launch and support features. In the spirit of DevOps, using all of these tools together for security analysis becomes just another task in your CI/CD pipeline.

For example, unit testing code paths through an application is common, so instead of just testing the code's functionality, let's audit the code from a security perspective. Taking time to work through your application and technology's custom use cases can help prevent costing your organization bad publicity, loss of customer faith, time, and money. Plan and test for the worst. In the beginning, implementing these concepts will have a certain amount of overhead (time and money), but the payoff, in terms of additional security assurance, is huge. As your organization's practices mature, the overhead associated with implementing DevOps in the Information Security space will improve.

Let's assume we are using these popular DevOps tools and start iterating on a security testing CI/CD pipeline:

- Docker
- Jenkins
- Ansible

For background, we use these tools everyday to help clients automate CI/CD pipelines for application development, and here's why we like them:

- Docker containers are great because they are a standalone, deployment-agnostic solution that can run on a laptop, on-premise, or in the cloud
- Jenkins is useful because it is so easy to set up complex, automated test jobs with almost every modern programming language
- Ansible's IT automation engine makes it easily to build a yaml playbook for cloud provisioning, configuration management, application deployment, intra-service orchestration, and many more

Additionally, all of these tools have commercially-supported counterparts that enterprises can choose to adopt. While the free, open-source versions work great, many large organizations find that commercial support gives them peace of mind that security patches will be delivered quickly, they are not dependent on community help, and that features of interest to the enterprise will be prioritized. Combining these three technologies makes application testing easy for small and large teams so why not use them for security too?

Defining Use Cases

Anyone interested in securing a web application or REST API should explore the OWASP (Open Web Application Security Project) website. Specifically, the OWASP Top 10 Application Security Risks post provides a great starting point for security testing guidelines by providing an initial set of 10 use cases to secure your applications, data, and customers.

For those new to security automation, the 2017–OWASP Top 10 Application Security Risks highlights internal, external, configuration, and intrusion use cases:

- A1 Injection
- A2 Broken Authentication
- A3 Sensitive Data Exposure
- A4 XML External Entities
- A5 Broken Access Control
- A6 Security Misconfiguration
- A7 Cross Site Scripting
- A8 Insecure Deserialization
- A9 Using Components with Known Vulnerabilities
- A10 Insufficient Logging and Monitoring

Approach – Analysis Techniques

While the OWASP Top 10 Cheat Sheet is great for laying out a roadmap, it doesn't show exactly how to analyze and find vulnerabilities or threats. As you start to set up your own testing harness, we would encourage you to incorporate both Static Application Security Testing (SAST) and Dynamic Application Security Testing (DAST) tools for increasing your testing footprint.

Most static testing tools scan code like any other static analysis tool, but the world of dynamic application testing is an arms race where previously-discovered attacks are thrown at your running application layers to see what breaks. DAST tools try to emulate hackers, so that by the time you have automated your DAST unit tests, you will have a better core understanding of your exposed surface area (and how to minimize it).

As time progresses, tools are updated and patched. We usually recommend setting up a quarterly review to confirm the DevOps tooling is still actively maintained (if you are not using a vendor-supported commercial solution) and checkout some of the potential new players in the space. The security testing ecosystem waits for no one.

Now let's discuss how to build a testing framework utilizing these tools that maximizes the considered use cases and minimizes the headache.

Building an OWASP Security Testing Harness

I built the repository below as a standalone docker container to perform static and dynamic OWASP security testing with Jenkins and Ansible. As a disclaimer, this repository is only for demonstration purposes. Please do not assume it will automatically secure and find all risks within your infrastructure and software. It takes time to properly secure software and just cloning this repository will not suffice for all use cases. It was last tested on 2018-01-18, and there are no guarantees it will not have issues in the future due to updates and fixes.

https://github.com/jay-johnson/owasp-jenkins

Under the hood, this single repository was built to cover most of the OWASP Top 10 for python runtimes to help demonstrate how to identify and qualify an application's Security Risks using FOSS tools. It also contains a sample Dependency Check pom file with most of the common language analyzers enabled by default (node.js, java, ruby, .NET, php, etc). This docker image contains the following capabilities:

1. Test Cases. Lots of test cases
    - One of the great free security testing resources is the NIST National Vulnerability Database (NVD). It is an actively-updated database of all known, publicly-released *Common Vulnerabilities and Exposures (CVEs)*. The NVD is essentially a list of all test cases to find bugs and potential risks that are in your applications. The container uses the NIST Data Mirror for downloading the NVD inside the container during the docker build step. For automation purposes you can just rebuild the container and get the latest CVEs automatically added into your CI/CD tooling.
2. Third Party Analysis–Dependency Checker
    - The OWASP community has a great ecosystem actively building dedicated tools for helping secure your apps. I included the Dependency Check tool to run through all the NVD CVE test cases to find evidence worth reviewing. It scans your applications' dependencies to find ones with known issues. With all the open source tools today it never hurts to check what third-party code you are using that might be doing something bad. Ideally all third-party modules or libraries are vendor-supported with a contract and undergo an internal code review whenever updates are brought into a build.
        i. We recommend setting up your CI/CD tools to run the Dependency Check before any Pull Request (PR) is merged into a master branch (change request, code update, or any other build testing phase). **Also, make sure it updates the NVD at least once a week.**
3. Static Application Security Testing
    - The container includes the python security analysis tool Openstack Bandit. Bandit is great because it comes with its own Vulnerability Tests plugins framework for helping roll your own security tests.
        i. We recommend setting up your CI/CD tools to run the static analysis before any PR is merged into a master branch or change request or code update and any other build testing phase.
4. Dynamic Application Security Testing
    - The container includes OWASP ZAPROXY (ZAP) which is a spider crawler that attacks a targeted service inside or outside the container and reports its findings. Also ZAP has a detailed coverage report on how it helps test most of the OWASP Top 10, and has a community-scripts repository for examples on rolling your own tests. DAST tools are usually something we run during integration tests because it can take some time to scan APIs with a large CRUD footprint. Tools like ZAP are also great for automating once a day against production for peace of mind to make sure nothing is broken.

Additionally, since these dynamic testing tools are actively crawling over a site or API, you can use them against any other server stacks too (node.js, java, ruby, .NET, php, etc.). Do you think hackers are going to use the same tech you are using to break into your infrastructure?

i. We recommend setting up your CI/CD tools to run the Dependency Check before any Pull Request (PR) is merged into a master branch (change request, code update, or any other build testing phase). Also, make sure it updates the NVD at least once a week.

Now that we have our tools, let's step back and review how this approach with these FOSS tools helps analyze our web applications and REST APIs for the 2017 OWASP Top 10.

OWASP Coverage Analysis–Harness Review

How does this container and approach help cover applications for the 2017 OWASP Top 10 Application Security Risks?

The container was built to help quickly secure python application code. There are tradeoffs to adding more and more tools to cover each item because while the tools might help find more security risks, the tradeoff is that your team now just added more testing overhead to support tools and keep them updated with your application builds. We do think that if your team can support this overhead, then it makes sense to add more analysis tooling to help catch more security issues before they get on production. The more tooling you have the more upkeep, but the benefits in the long run are enormous.

This is a table to visualize how the tools help test the OWASP Top 10:

OWASP Top 10 Application Security Risks in 2017	DAST	SAST	Third Party
A1 Injection	ZAP	Bandit	Dependency Check
A2 Broken Authentication	ZAP	Bandit	Dependency Check
A3 Sensitive Data Exposure	ZAP	Bandit	Dependency Check
A4 XML External Entities	ZAP	Bandit	Dependency Check
A5 Broken Access Control	ZAP	Bandit	Dependency Check
A6 Security Misconfiguration	ZAP	Bandit	Dependency Check
A7 Cross Site Scripting	ZAP	Bandit	Dependency Check
A8 Insecure Deserialization	None	Bandit	Dependency Check
A9 Using Components with Known Vulnerabilities	ZAP	None	Dependency Check
A10 Insufficient Logging and Monitoring	None	None	None

Automation

The container includes a pre-configured Jenkins server with a dummy, self-signed X509 certificate (and private key) for setting up automated jobs with snippets for each job inside the README. We recommend replacing the included cert and key with ones that meet your organization's PKI standards. These snippets use the corresponding Ansible playbook to run the analysis job and write the results to a file. This file is then attached to a test success event and emailed to the team for review. Additionally, because the testing harness runs using Ansible playbooks, you can automate scanning any repository or code without even logging into Jenkins at all.

Want to generate an OWASP HTML Report for third-party vulnerabilities on some of the most popular repositories on GitHub using the Dependency Check tool?

Note these HTML reports are also available in the repository:

https://github.com/jay-johnson/security-testing-reports

Scan Django

Scan Django and generate an OWASP HTML Report for third-party vulnerabilities:

```
repo=https://github.com/django/django.git
docker run --name owasp-jenkins -p 8443:8443 -v $(pwd):/opt/reports
-it -d jayjohnson/owasp-jenkins:latest && docker exec -it owasp-
jenkins git clone $repo /opt/scanrepo && docker exec -it owasp-jenkins
ansible-playbook -i inventories/inventory_dev run-owasp-analysis.yml
-e owasp_scan_dir="/opt/scanrepo" -e owasp_report_file="/opt/reports/
owasp-django-report-$(date +'%Y-%m-%d-%H-%M-%S').html" && docker stop
owasp-jenkins && docker rm owasp-jenkins
ls -l owasp-django-report-*.html
```

View Archived Version on GitHub

Scan React

Scan React and generate an OWASP HTML Report for third-party vulnerabilities:

```
repo=https://github.com/facebook/react.git
docker run --name owasp-jenkins -p 8443:8443 -v $(pwd):/opt/reports
-it -d jayjohnson/owasp-jenkins:latest && docker exec -it owasp-
jenkins git clone $repo /opt/scanrepo && docker exec -it owasp-jenkins
ansible-playbook -i inventories/inventory_dev run-owasp-analysis.yml
-e owasp_scan_dir="/opt/scanrepo" -e owasp_report_file="/opt/reports/
owasp-react-report-$(date +'%Y-%m-%d-%H-%M-%S').html" && docker stop
owasp-jenkins && docker rm owasp-jenkins
ls -l owasp-react-report-*.html
```

View Archived Version on GitHub

Scan Vue

Scan Vue and generate an OWASP HTML Report for third-party vulnerabilities:

```
repo=https://github.com/vuejs/vue.git
docker run --name owasp-jenkins -p 8443:8443 -v $(pwd):/opt/reports
-it -d jayjohnson/owasp-jenkins:latest && docker exec -it owasp-
jenkins git clone $repo /opt/scanrepo && docker exec -it owasp-jenkins
ansible-playbook -i inventories/inventory_dev run-owasp-analysis.yml
-e owasp_scan_dir="/opt/scanrepo" -e owasp_report_file="/opt/reports/
owasp-vue-report-$(date +'%Y-%m-%d-%H-%M-%S').html" && docker stop
owasp-jenkins && docker rm owasp-jenkins
ls -l owasp-vue-report-*.html
```

View Archived Version on GitHub

Scan Angular

Scan Angular and generate an OWASP HTML Report for third-party

```
repo=https://github.com/angular/angular.git
docker run --name owasp-jenkins -p 8443:8443 -v $(pwd):/opt/reports
-it -d jayjohnson/owasp-jenkins:latest && docker exec -it owasp-
jenkins git clone $repo /opt/scanrepo && docker exec -it owasp-jenkins
ansible-playbook -i inventories/inventory_dev run-owasp-analysis.yml
-e owasp_scan_dir="/opt/scanrepo" -e owasp_report_file="/opt/reports/
owasp-angular-report-$(date +'%Y-%m-%d-%H-%M-%S').html" && docker stop
owasp-jenkins && docker rm owasp-jenkins
ls -l owasp-angular-report-*.html
```

View Archived Version on GitHub

Scan Ruby on Rails

Scan Ruby on Rails and generate an OWASP HTML Report for third-party vulnerabilities:

```
repo=https://github.com/rails/rails
docker run --name owasp-jenkins -p 8443:8443 -v $(pwd):/opt/reports
-it -d jayjohnson/owasp-jenkins:latest && docker exec -it owasp-
jenkins git clone $repo /opt/scanrepo && docker exec -it owasp-jenkins
ansible-playbook -i inventories/inventory_dev run-owasp-analysis.yml
-e owasp_scan_dir="/opt/scanrepo" -e owasp_report_file="/opt/reports/
owasp-ror-report-$(date +'%Y-%m-%d-%H-%M-%S').html" && docker stop
owasp-jenkins && docker rm owasp-jenkins
ls -l owasp-ror-report-*.html
```

View Archived Version on GitHub

Scan Shadowsocks Windows

Scan Shadowsocks Windows and generate an OWASP HTML Report for third-party vulnerabilities:

```
repo=https://github.com/shadowsocks/shadowsocks-windows.git
docker run --name owasp-jenkins -p 8443:8443 -v $(pwd):/opt/reports
-it -d jayjohnson/owasp-jenkins:latest && docker exec -it owasp-
jenkins git clone $repo /opt/scanrepo && docker exec -it owasp-jenkins
ansible-playbook -i inventories/inventory_dev run-owasp-analysis.yml
-e owasp_scan_dir="/opt/scanrepo" -e owasp_report_file="/opt/reports/
owasp-shadowsockswindows-report-$(date +'%Y-%m-%d-%H-%M-%S').html" &&
docker stop owasp-jenkins && docker rm owasp-jenkins
ls -l owasp-shadowsockswindows-report-*.html
```

View Archived Version on GitHub

Scan Laravel

Scan Laravel and generate an OWASP HTML Report for third-party vulnerabilities:

```
repo=https://github.com/laravel/laravel
docker run --name owasp-jenkins -p 8443:8443 -v $(pwd):/opt/reports
-it -d jayjohnson/owasp-jenkins:latest && docker exec -it owasp-
jenkins git clone $repo /opt/scanrepo && docker exec -it owasp-jenkins
ansible-playbook -i inventories/inventory_dev run-owasp-analysis.yml
-e owasp_scan_dir="/opt/scanrepo" -e owasp_report_file="/opt/reports/
owasp-laravel-report-$(date +'%Y-%m-%d-%H-%M-%S').html" && docker stop
owasp-jenkins && docker rm owasp-jenkins
ls -l owasp-laravel-report-*.html
```

View Archived Version on GitHub

Scan Django REST Framework

Scan Django REST Framework and generate an OWASP HTML Report for third-party vulnerabilities:

```
repo=https://github.com/encode/django-rest-framework.git
docker run --name owasp-jenkins -p 8443:8443 -v $(pwd):/opt/reports
-it -d jayjohnson/owasp-jenkins:latest && docker exec -it owasp-
jenkins git clone $repo /opt/scanrepo && docker exec -it owasp-jenkins
ansible-playbook -i inventories/inventory_dev run-owasp-analysis.yml
-e owasp_scan_dir="/opt/scanrepo" -e owasp_report_file="/opt/reports/
owasp-drf-report-$(date +'%Y-%m-%d-%H-%M-%S').html" && docker stop
owasp-jenkins && docker rm owasp-jenkins
ls -l owasp-drf-report-*.html
```

View Archived Version on GitHub

Want to generate Bandit reports for some of the most popular python projects?

Scan Tensorflow and generate a Bandit HTML report:

```
repo=https://github.com/tensorflow/tensorflow.git
docker run --name owasp-jenkins -p 8443:8443 -v $(pwd):/opt/reports
-it -d jayjohnson/owasp-jenkins:latest && docker exec -it owasp-
jenkins git clone $repo /opt/scanrepo && docker exec -it owasp-jenkins
ansible-playbook -i inventories/inventory_dev run-bandit-analysis.yml
-e bandit_scan_dir="/opt/scanrepo" -e bandit_report_file="/opt/reports/
bandit-tf-report-$(date +'%Y-%m-%d-%H-%M-%S').html" && docker stop
owasp-jenkins && docker rm owasp-jenkins
ls -l bandit-tf-report-*.html
```

View Archived Version on GitHub

Scan Flask and generate a Bandit HTML report:

```
repo=https://github.com/pallets/flask.git
docker run --name owasp-jenkins -p 8443:8443 -v $(pwd):/opt/reports
-it -d jayjohnson/owasp-jenkins:latest && docker exec -it owasp-
jenkins git clone $repo /opt/scanrepo && docker exec -it owasp-jenkins
ansible-playbook -i inventories/inventory_dev run-bandit-analysis.yml
-e bandit_scan_dir="/opt/scanrepo" -e bandit_report_file="/opt/reports/
bandit-flask-report-$(date +'%Y-%m-%d-%H-%M-%S').html" && docker stop
owasp-jenkins && docker rm owasp-jenkins
ls -l bandit-flask-report-*.html
```

View Archived Version on GitHub

Scan Ansible and generate a Bandit HTML Report:

```
repo=https://github.com/ansible/ansible.git
docker run --name owasp-jenkins -p 8443:8443 -v $(pwd):/opt/reports
-it -d jayjohnson/owasp-jenkins:latest && docker exec -it owasp-
jenkins git clone $repo /opt/scanrepo && docker exec -it owasp-jenkins
ansible-playbook -i inventories/inventory_dev run-bandit-analysis.yml
-e bandit_scan_dir="/opt/scanrepo" -e bandit_report_file="/opt/reports/
bandit-ab-report-$(date +'%Y-%m-%d-%H-%M-%S').html" && docker stop
owasp-jenkins && docker rm owasp-jenkins
ls -l bandit-ab-report-*.html
```

View Archived Version on GitHub

So it's easy to run OWASP scans and analyze threats. So how do we keep from being the next major hack?

Adoption and Staying Vigilant

As 2018 kicks off, it's pretty obvious security is a never-ending arms race, and like everything in technology the approach and process discussed here could be obsolete over the coming years. Understanding how your application communicates with internal and the external world are critical to knowing how to secure it. Updates to toolchains, OWASP, and the NIST NVD require setting up update jobs, but reviewing what these tools find still requires an organization to want to adopt a process to secure your business. Security testing is not 100% automated like unit testing. Even the best gates will rust.

Coming Soon and Conclusion

Here's some topics we'll be taking a look at in an upcoming report:

- Malware Integration Testing–component resiliency testing using contained threats
- Runtime Hardening–customizing the application runtime
- Network Analysis–transports, protocols, firewall and dns

In conclusion, thanks for reading and let us know how your security strategy fits into this approach. If you organization has an interest in using devops tools to automate security tests then please reach out to us at Levvel and we can get you started. Lastly, let us know which of these sound interesting and we might queue up a report on how we can tackle it:

- Privilege Escalation–using kernel or container exploits to become a switch users (like root)
- API Usage Analysis–traffic modeling with AI/ML to identify bad API actors
- Penetration Testing–post-build, deployment intrusion testing

About Levvel

We are an IT, strategy, and design consulting firm that combines the innovative DNA of a startup with the wisdom, scalability, and process rigor of a Fortune 100 company.

Learn more about how Levvel can support your DevOps strategy goals by contacting us at hello@levvel.io.