Bright

BRIGHT SECURITY IDE EXTENSION FOR DEVELOPER-CENTRIC DAST SCANS

www.brightsec.com

The need to protect software resources against bad actors increases year over year. As the digital world continues to evolve, the ability to gain financial profit, or cause damage through cyber attacks increases. Customers expect organizations to proactively identify and remediate vulnerabilities to protect their data." Many standards (e.g., ISO/IEC, SOC, etc.) require organizations to identify cyber threats actively using the appropriate security controls.

Traditional AppSec practices await for all development and functional testing activities to complete before the security validation starts. This leads to vulnerabilities being identified very late in the release process. In fact, due to businesses' need to release the software to the end users, companies release software with known vulnerabilities, as they cannot postpone the release until all, or even all high and critical issues are resolved. Companies try to manage risk by assuming exploitation likelihood instead of fixing the open attack vectors.

Bright Security's Dev-Centric DAST solution resolves this challenge by enabling vulnerability-finding in the development process. Bright provides an extension to the developer IDE (Integrated Development Environment) so developers can invoke security testing within their working environment and as part of the dev process without the need for a supplementary UI (e.g., integrated extension into Visual Studio Code). Members of the development team can complete their service development and immediately run an attack simulation to validate that the new code does not expose an attack vector. It can be part of the Pull Request (PR) process or alongside this process. The security test findings are also shown both within the IDE, with detailed proof of the vulnerability and remediation guidelines so developers can take immediate action, and within the Bright UI so AppSec teams can learn, track and continuously improve. In addition, the DevSecOps team can add additional critical scans to the CI/CD pipeline to ensure the new build is secure.

Adopting a Dev-Centric DAST approach drives significant value for organizations

- Identify vulnerabilities early in the development cycle when there is time to remediate and avoid pushing known vulnerabilities to production thus significantly reducing risk.
- Remediating a vulnerability early in the SDLC can be up to 60X faster than remediating it in pre-prod, or production.
- Providing developers with validated vulnerabilities saves them time chasing ghosts (unvalidated vulnerabilities) and wasting time on false positives.
- The developers are not required to learn a new tool or have a context switch - the DAST capabilities are available within their day-to-day working environment.
- Improved collaboration, trust and transparency between the developers team and the AppSec team.
- Executive visibility for the application security posture as it is being developed.

How does Dev-Centric DAST work?

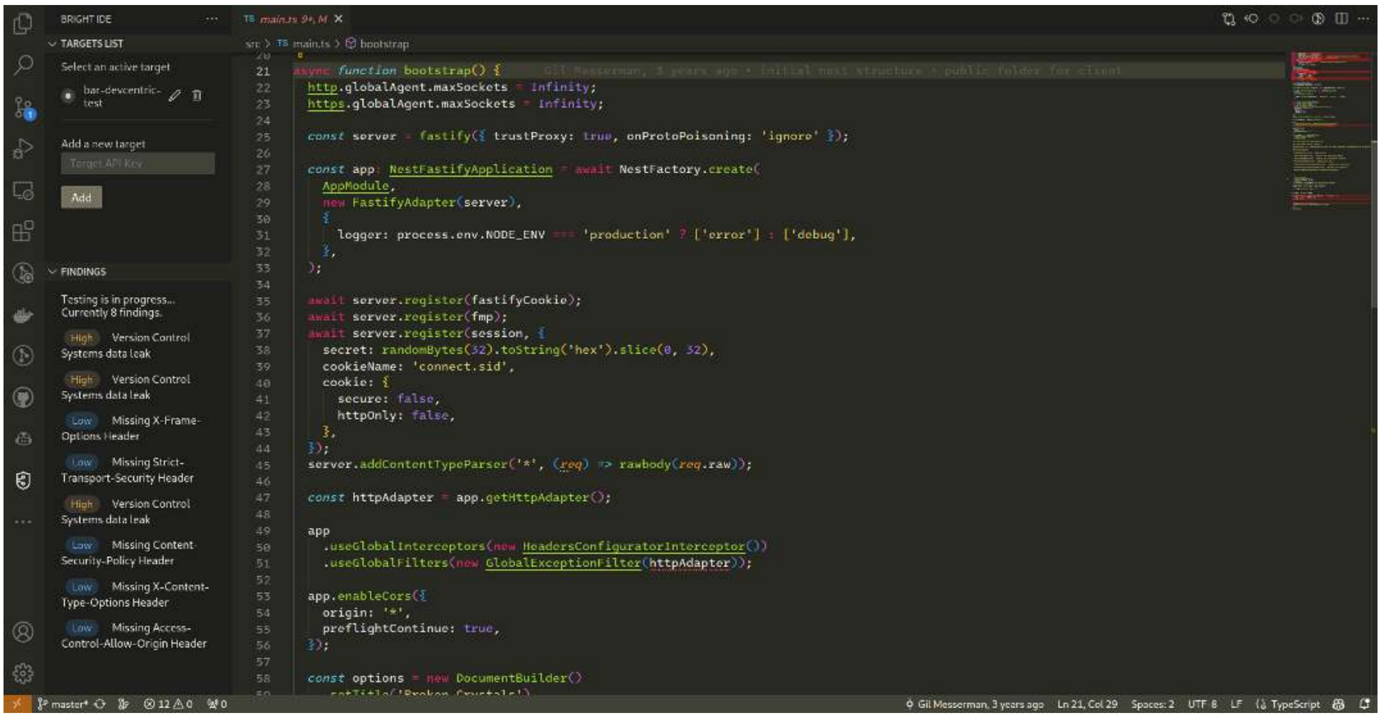
Bright’s DAST identifies the attack surface - all the entry points of the application, microservices, or APIs that might be used as exploitation targets. This is achieved by running a discovery based on Bright's automated crawler or pointing to an API Schema like OAS for REST or introspection for GraphQL. The discovery can be done centrally by the AppSec team or individually by the developers within the IDE. Once the attack surface is available, a developer can select the specific entry point which they wish to simulate an attack and execute the tests. Bright provides a recommended set of tests for each stage of the pipeline, and the developer can also override it for their specific needs.

The tests involve a series of attack simulations where Bright manipulates the requests sent to the application, similar to what an actual attacker or penetration tester would do. They then validate whether the response is as expected or if the manipulation of the request was successful. If the response reveals unplanned information, such as personally identifiable information (PII) or data that should not be exposed to attackers, the application is considered vulnerable.

At the end of the testing cycle, all vulnerability findings are shown to the developer in the IDE with the proof of vulnerability and the exact actions that were performed to find this open attack vector. The AppSec team can also monitor the findings from the main Bright dashboard. The findings are shown by their severity to simplify the prioritization of the remediation process. Once the developer fixes the issue, he can re-run the specific test that found the vulnerability to validate his fix (at this point, the concept is similar to test-driven development methodology (TDD)

Adopting this methodology dramatically improves the application's security posture and gives different stakeholders the confidence to release secure working software at high velocity.

Example for Bright findings in the left panel:

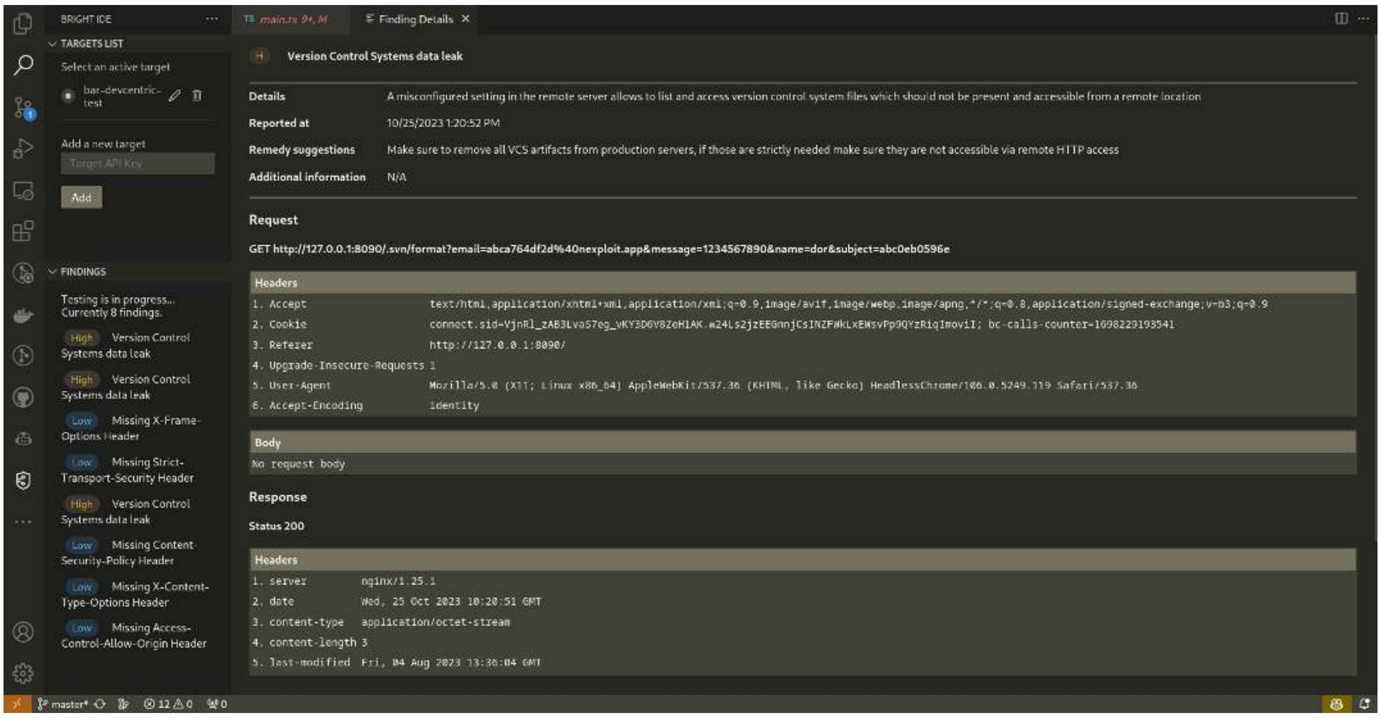


Testing is in progress...
Currently 8 findings.

- High Version Control Systems data leak
- High Version Control Systems data leak
- Low Missing X-Frame-Options Header
- Low Missing Strict-Transport-Security Header
- High Version Control Systems data leak
- Low Missing Content-Security-Policy Header
- Low Missing X-Content-Type-Options Header
- Low Missing Access-Control-Allow-Origin Header

```
async function bootstrap() {      Gil Messerman, 3 years ago • initial nest structure + public folder for client
  http.globalAgent.maxSockets = Infinity;
  https.globalAgent.maxSockets = Infinity;

  const server = fastify({ trustProxy: true, onProtoPoisoning: 'ignore' });

  const app: NestFastifyApplication = await NestFactory.create(
    AppModule,
    new FastifyAdapter(server),
    {
      logger: process.env.NODE_ENV === 'production' ? ['error'] : ['debug'],
    },
  );

  await server.register(fastifyCookie);
  await server.register(fmp);
  await server.register(session, {
    secret: randomBytes(32).toString('hex').slice(0, 32),
    cookieName: 'connect.sid',
    cookie: {
      secure: false,
      httpOnly: false,
    },
  });
  server.addContentTypeParser('*', (req) => rawbody(req.raw));

  const httpAdapter = app.getHttpAdapter();

  app
    .useGlobalInterceptors(new HeadersConfiguratorInterceptor())
    .useGlobalFilters(new GlobalExceptionFilter(httpAdapter));

  app.enableCors({
    origin: '*',
    preflightContinue: true,
  });

  const options = new DocumentBuilder()
    .setTitle('Broken Crystals')
```

Example for Bright’s proof of vulnerability:



Testing is in progress...
Currently 8 findings.

- High Version Control Systems data leak
- High Version Control Systems data leak
- Low Missing X-Frame-Options Header
- Low Missing Strict-Transport-Security Header
- High Version Control Systems data leak
- Low Missing Content-Security-Policy Header
- Low Missing X-Content-Type-Options Header
- Low Missing Access-Control-Allow-Origin Header

H Version Control Systems data leak

Details	A misconfigured setting in the remote server allows to list and access version control system files which should not be present and accessible from a remote location
Reported at	10/25/2023 1:20:52 PM
Remedy suggestions	Make sure to remove all VCS artifacts from production servers, if those are strictly needed make sure they are not accessible via remote HTTP access
Additional information	N/A

Request

GET http://127.0.0.1:8090/.svn/format?email=abca764df2d%40nexploit.app&message=1234567890&name=dor&subject=abc0eb0596e

Headers

```
1. Accept                    text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
2. Cookie                    connect.sid=VjnRl_zAB3LvaS7eg_vKY3D6V8ZeH1AK.w24Ls2jzEEGnnjCsINZFWkLxEWsvPp9QYzRiqImoviI; bc-calls-counter=1698229193541
3. Referer                   http://127.0.0.1:8090/
4. Upgrade-Insecure-Requests 1
5. User-Agent                Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) HeadlessChrome/106.0.5249.119 Safari/537.36
6. Accept-Encoding           identity
```

Body

```
No request body
```

Response

Status 200

Headers

```
1. server         nginx/1.25.1
2. date           Wed, 25 Oct 2023 10:20:51 GMT
3. content-type   application/octet-stream
4. content-length 3
5. last-modified  Fri, 04 Aug 2023 13:30:04 GMT
```