intrinsic

Intrinsic: A principled security product

Intrinsic's goal is to provide principled security at the application layer without hindering productivity. To do this we've spent years in research figuring out how to make enforcement security easy and practical enough for anyone to use.

Intrinsic is a Node.js library that wraps applications. Application developers write policy files whitelisting the privileges of an application and Intrinsic enforces that the application abides by them. Policies are fine-grained and give access to resources such as the network, file system, child process, databases, and more. To learn more about the product, visit
https://intrinsic.com/product

Intrinsic protects against against attacks allowed common developer mistakes to complex unknown vulnerabilities. Here are some examples on how Intrinsic protects three common attack vectors: Remote Code Execution, SQL Injection, and Path Traversal attacks.

Remote Code Execution

In the event that a malicious actor is able to execute code in your application, Intrinsic has you covered. There are many ways an attacker can execute malicious code in your applications. A popular vector today is through malicious modules. As we've seen in the eslint-scope and event-stream attacks, hijacking modules poses an immense threat to applications today. You can read more about it here: https://medium.com/p/863ae949e7e8.

Let's take an example of an application that makes requests to many different services through API calls. There's a route `/chargeUser` that charges a customer using PayPal. Suppose the application stores the PayPal key as a hard coded value. The application also stores other APIs keys for Google Vision, Twitter, and many more as environment variables.

Now let's say an attacker hijacks an npm module deep within your dependency tree. This module now includes malicious code that intercepts all HTTP calls and mirrors those calls to an evil server. This code does not break any functionality in the code and employs a variety of mechanisms to remain undetected.

The malicious code is now stealing the PayPal API key. It also reads the application's environment variables and send them to its evil server through an HTTP call. It has successfully stolen all the application's API keys.

If the application was using Intrinsic, the malicious code would be neutralized and unable to steal any API keys because its HTTP calls to the evil server would be blocked.

The following Intrinsic policy could be applied to the `/chargeUser` route:

```
routes.post('/chargeUser', policy => {
  policy.outboundHttp.allowPost('https://api.paypal.com/v1/payments/payment');
});
```

With the policy set above, the route `/chargeUser` is only able to call the specific PayPal endpoint needed. If the application tries to make any other HTTP call, it will be blocked. Therefore, the malicious code that was introduced by a hijacked module is now incapable of making the HTTP call to its evil server. The application will log an attempted HTTP call as a policy violation so that developers and security engineers can take action.

SQL Injections

Intrinsic protects applications from SQL injections by use of database policies. By restricting how an application can access the database any unexpected queries are automatically blocked.

Let’s take the following vulnerable code:

```
let query = `SELECT email FROM users WHERE "type" = '${req.param('userType')}'`;
```

If an attacker passes in `admin' OR 1=1 --` as the userType then the query will return all the emails from the table.

With Intrinsic, we are able to protect the application even with this vulnerable code. Take the following policy database policies:

```
policy.postgresql.allowQuery(databaseName,
  `SELECT email FROM users WHERE "type" = 'user'`);
policy.postgresql.allowQuery(databaseName,
  `SELECT email FROM users WHERE "type" = 'guest'`);
```

This policy enforces that the this exact query is only allowed and nothing else. Therefore when an attacker attempts an SQL injection the query will be blocked, even though the application’s code is vulnerable to it.

Path Traversals

Intrinsic protects against path traversal attacks by blocking unprivileged access to unexpected areas. For example, assume we have an API server where a blog route returns static files from a directory called `public`

If the user makes the following call: `GET https://something.com/blog/index.html` the application returns the static `public/index.html` file. The application takes in the filename as a URL parameter which it uses to return the requested file.

The vulnerable code looks like this:

```
path.join('./public', req.filePath);
```

With this vulnerability an attacker can now pass in `../secrets/file.txt` as the filePath parameter and the application will return the secret file.

Intrinsic protects the application by denying access to any file in the system other than the specific files whitelisted. For the above route, here’s an example policy set:

```
routes.get('blog/**', policy => {
  policy.fs.allowRead(`${__dirname}/public/*`);
});
```

This policy would only allow files in the public directory to be read by the blog route. When the attacker tries to access a file outside of the public directory, Intrinsic blocks the access.