True Code

Developer tooling to get security right.

Driving secure development efficiency through collaboration & automation

riscure

WHY WE STARTED IT

Security vulnerabilities in software have led to numerous exploits in the last years. The fact that the size of software running on devices is becoming bigger and bigger as well as the number of use cases that need to be supported only makes it more likely that future exploits will increase.

To prevent hacks that bring down customer trust or can cause revenue loss because of piracy and are costly to mitigate after product releases, software needs to be evaluated.

Up until now, the best evaluation process is a highly manual task with a security expert, which results in high costs and long lead times. It is also quite common that an evaluation takes place at the end of the development cycle causing up to 100x higher costs to resolve issues, as opposed to when an issue would have been found in the development phase.

Separate from a manual evaluation, there are also automated code check tools. However, from what we see in many manual evaluations, the tools being used focus on code quality standards, do not find the critical problems and often report irrelevant ones.

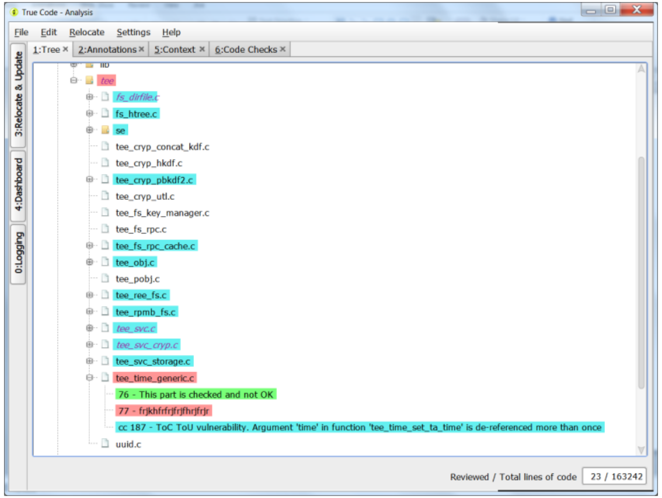
THE PRODUCT

We developed **True Code**.

A tool purely focused on finding security vulnerabilities in source code and enabling natural collaboration between security evaluators and the development team to discover vulnerabilities as early as possible and boost efficiency to resolve issues.

True Code Analysis

RISCURE



```
lib
tee
  fs_dirfile.c
  fs_htree.c
  se
  tee_cryp_concat_kdf.c
  tee_cryp_hkdf.c
  tee_cryp_pbkdf2.c
  tee_cryp_utl.c
  tee_fs_key_manager.c
  tee_fs_rpc.c
  tee_fs_rpc_cache.c
  tee_obj.c
  tee_pobj.c
  tee_ree_fs.c
  tee_rpmb_fs.c
  tee_svc.c
  tee_svc_cryp.c
  tee_svc_storage.c
  tee_time_generic.c
    76 - This part is checked and not OK
    77 - frjkhfrfrjfrjfhrjfrjr
    cc 187 - ToC ToU vulnerability. Argument 'time' in function 'tee_time_set_ta_time' is de-referenced more than once
  uuid.c
```

Reviewed / Total lines of code 23 / 163242

- ✓ Save costs by finding vulnerabilities early in the development process and solve issues in the most efficient way
- ✓ Combine manual and automated reviews
- ✓ Easy integration in your software development lifecycle
- ✓ Security experts and developers both work in the same environment
- ✓ Integrated in Eclipse or stand-alone to use with any IDE of choice

THE CHALLENGES IT SOLVES

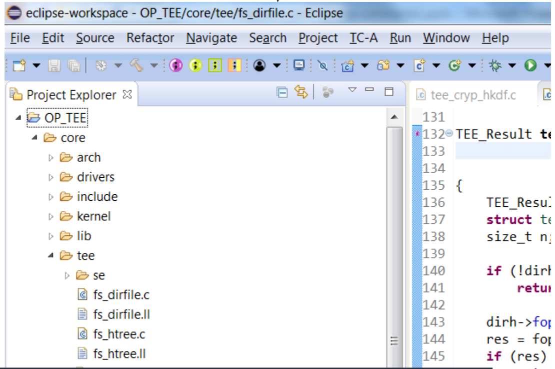
Save time and costs

Finding vulnerabilities and issues during the development phase and immediately resolving them can be up to a 100 times cheaper compared to doing the same later in the process. True Code brings this promise within reach through a tight integration in the development, sharing found vulnerabilities instantly with all team members. True Code integrates tightly with the development environment that is used by your team and integrates with any other SLD tools to automate as much as possible

Combine expert knowledge and automation

Expert knowledge is needed to find vulnerabilities in a code base. We have used years of experience obtained during manual code reviews to strengthen the automated vulnerability finding capabilities of True Code so that it finds 'real' issues instead of false positives. Next to that we strongly believe that to achieve the highest level of security a combination of automated checks and expert manual code review gives the best results. In order to achieve this, both kinds of checks are done from the same platform and this is also the platform used by the developers. This encourages collaboration between all teams working together in delivering the product.

True code options easily available and integrated in the IDE


eclipse-workspace - OP_TEE/core/tee/fs_dirfile.c - Eclipse
File Edit Source Refactor Navigate Search Project TC-A Run Window Help
Project Explorer
OP_TEE
core
arch
drivers
include
kernel
lib
tee
se
fs_dirfile.c
fs_dirfile.ll
fs_htree.c
fs_htree.ll
tee_cryp_hkdf.c

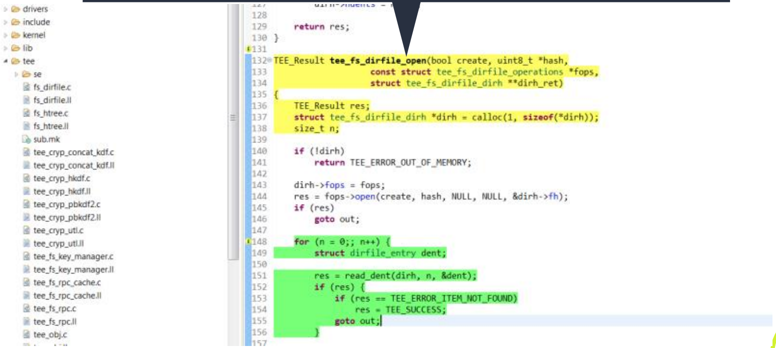
```
TEE_Result te

{
    TEE_Resu
    struct te
    size_t n

    if (!dirh
        retu

    dirh->fop
    res = fop
    if (res)
        goto
```

Color schemes help to easily identify different types of found vulnerabilities for everyone in the team



```

    return res;
}

TEE_Result tee_fs_dirfile_open(bool create, uint8_t *hash,
                               const struct tee_fs_dirfile_operations *fops,
                               struct tee_fs_dirfile_dirh **dirh_ret)
{
    TEE_Result res;
    struct tee_fs_dirfile_dirh *dirh = calloc(1, sizeof(*dirh));
    size_t n;

    if (!dirh)
        return TEE_ERROR_OUT_OF_MEMORY;

    dirh->fops = fops;
    res = fops->open(create, hash, NULL, NULL, &dirh->fh);
    if (res)
        goto out;

    for (n = 0;; n++) {
        struct dirfile_entry dent;

        res = read_dent(dirh, n, &dent);
        if (res) {
            if (res == TEE_ERROR_ITEM_NOT_FOUND)
                res = TEE_SUCCESS;
            goto out;
        }
```

THE CHALLENGES IT SOLVES

Context driven to reduce false positives

Context is a center point in True Code. For efficient checking of security vulnerabilities, context, basically a way to flag certain parts of your code base, is necessary. True Code allows you to define as much context as you need and has a rule based system to help you.

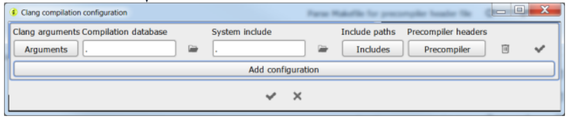
After context definition you can choose to run a code check on 1 specific context, all defined context or a subset.

Easy to configure

Setting up and configuring a code analysis tool can be a real burden for a development team. We have made this as smooth a possible, guaranteeing a minimal amount of effort to get started and a fast learning curve for the users.

To run the automated code checks, True Code will need to compile your sources, but providing a compilation database is sufficient to direct True Code to execute this compilation step. To start working with the collaboration features: choose your database , configure True Code to make use of this and you are good to go!

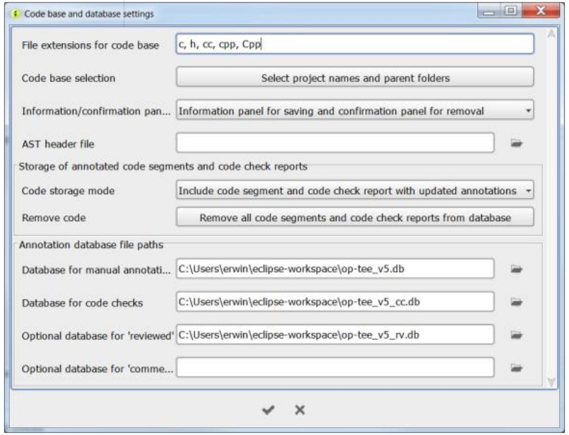
Setting up prerequisites for compilation needed for the automated checks is easy and straight forward



Clang compilation configuration

Clang arguments: Arguments
Compilation database: .
System include: .
Include paths: Includes
Precompiler headers: Precompiler

Set your code base to start working with True Code



Code base and database settings

File extensions for code base: c, h, cc, cpp, Cpp
Code base selection: Select project names and parent folders
Information/confirmation pan...: Information panel for saving and confirmation panel for removal
AST header file:

Storage of annotated code segments and code check reports
Code storage mode: Include code segment and code check report with updated annotations
Remove code: Remove all code segments and code check reports from database

Annotation database file paths
Database for manual annotati...: C:\Users\erwin\eclipse-workspace\op-tee_v5.db
Database for code checks: C:\Users\erwin\eclipse-workspace\op-tee_v5_cc.db
Optional database for 'reviewed': C:\Users\erwin\eclipse-workspace\op-tee_v5_rv.db
Optional database for 'comme...:

Choose the database that you want to choose for storing found vulnerabilities

THE CHALLENGES IT SOLVES

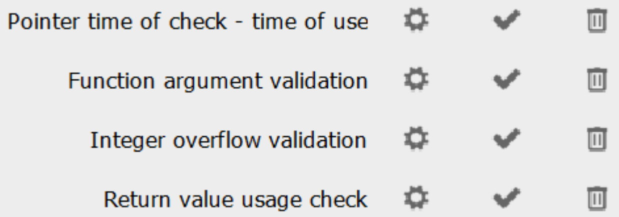
Enabling ‘live code reviews’

Manual code reviews are usually planned at the end of the development process. This means that solving found issues will be done long after the problem was introduced. True Code solves this issue by supporting ‘live reviews’. In this way, the evaluation team can work along with the development team while True Code helps keeping an overview which parts of the code have been reviewed and where the development team have made changes. This will make sure that the development team as well as the evaluation team both work as efficient as possible with review & fix time savings up to 30%.

Make extensive reporting obsolete

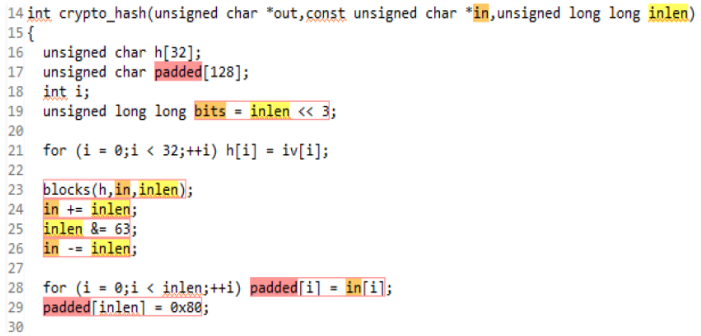
Reporting can consume a lot of time from the evaluation team. That’s why True Code keeps track of all issues, as well as progress with regard to solving them, in a database. Obviously this database can be queried with any SQL tool available, but True Code also has an option to generate a report based in database content. This saves valuable time from the evaluation team that instead can focus on security issues.

Each code check can have its own specific configuration making sure that the check runs as efficient as possible



Pointer time of check - time of use ⚙ ✓ 🗑
Function argument validation ⚙ ✓ 🗑
Integer overflow validation ⚙ ✓ 🗑
Return value usage check ⚙ ✓ 🗑

Dataflow visualization helps to more easily identify security vulnerabilities when manually checking



```
int crypto_hash(unsigned char *out,const unsigned char *in,unsigned long long inlen)
{
  unsigned char h[32];
  unsigned char padded[128];
  int i;
  unsigned long long bits = inlen << 3;

  for (i = 0;i < 32;++i) h[i] = iv[i];

  blocks(h,in,inlen);
  in += inlen;
  inlen &= 63;
  in -= inlen;

  for (i = 0;i < inlen;++i) padded[i] = in[i];
  padded[inlen] = 0x80;

```

UNIQUE FEATURES

Save costs and reduce time to market

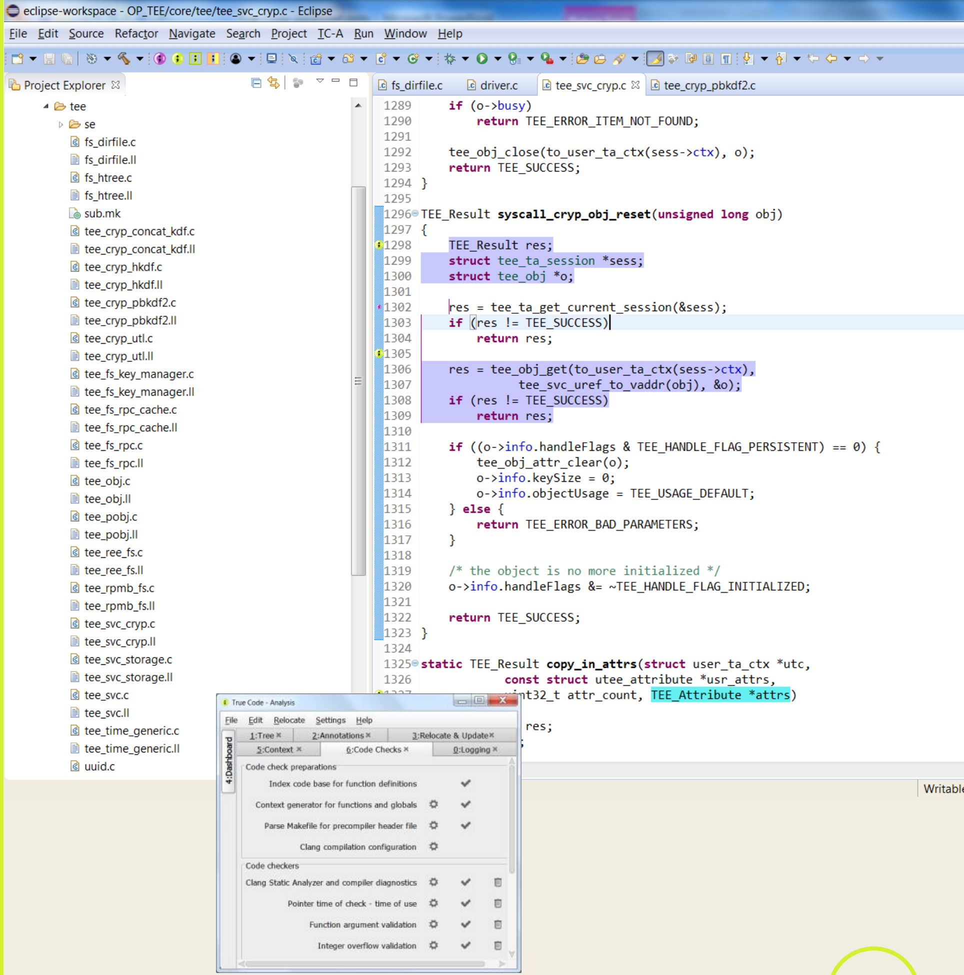
True code makes sure that you find vulnerabilities during development. Next to the automated checks that can be executed on a daily bases, True Code also facilitates collaboration with security experts in the development phase. Reducing (certification) costs and allowing you to reach your goals faster.

Fault injection vulnerability checks

Fault injection proves to be a method that is used often by attackers. True Code indicates specific vulnerabilities in source code related to fault injection

A pure security product

Many of the static code analysis products out in the market focus on a lot of things that might be of interest to a development team. Not True Code. True code is made for security purposes only with people who have a long track record in code evaluations and excel in security expertise. From collaboration to automated checks… it is intended to be the best at security, period.



```
	if (o->busy)
		return TEE_ERROR_ITEM_NOT_FOUND;

	tee_obj_close(to_user_ta_ctx(sess->ctx), o);
	return TEE_SUCCESS;
}

TEE_Result syscall_cryp_obj_reset(unsigned long obj)
{
	TEE_Result res;
	struct tee_ta_session *sess;
	struct tee_obj *o;

	res = tee_ta_get_current_session(&sess);
	if (res != TEE_SUCCESS)
		return res;

	res = tee_obj_get(to_user_ta_ctx(sess->ctx),
			  tee_svc_uref_to_vaddr(obj), &o);
	if (res != TEE_SUCCESS)
		return res;

	if ((o->info.handleFlags & TEE_HANDLE_FLAG_PERSISTENT) == 0) {
		tee_obj_attr_clear(o);
		o->info.keySize = 0;
		o->info.objectUsage = TEE_USAGE_DEFAULT;
	} else {
		return TEE_ERROR_BAD_PARAMETERS;
	}

	/* the object is no more initialized */
	o->info.handleFlags &= ~TEE_HANDLE_FLAG_INITIALIZED;

	return TEE_SUCCESS;
}

static TEE_Result copy_in_attrs(struct user_ta_ctx *utc,
			const struct utee_attribute *usr_attrs,
			nt32_t attr_count, TEE_Attribute *attrs)

	res;
```

Code check preparations

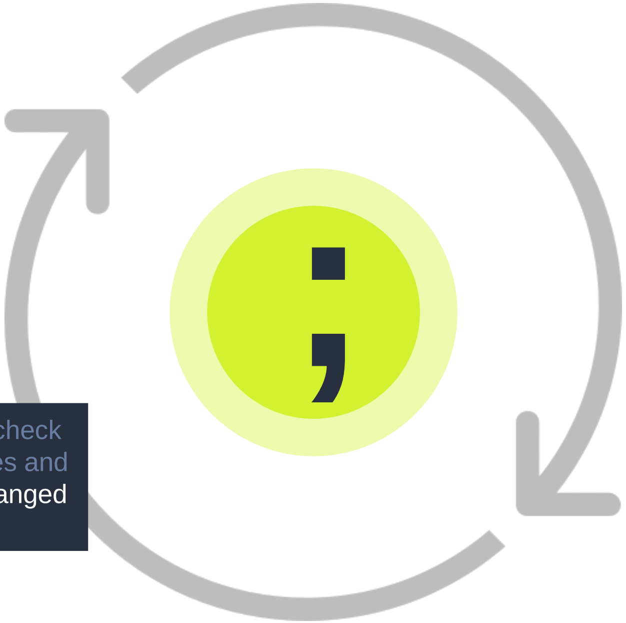
- Index code base for function definitions ✓
- Context generator for functions and globals ✓
- Parse Makefile for precompiler header file ✓
- Clang compilation configuration

Code checkers

- Clang Static Analyzer and compiler diagnostics ✓
- Pointer time of check - time of use ✓
- Function argument validation ✓
- Integer overflow validation ✓

USE CASE : COLLABORATION

True code keeps track of code that is developed by the team

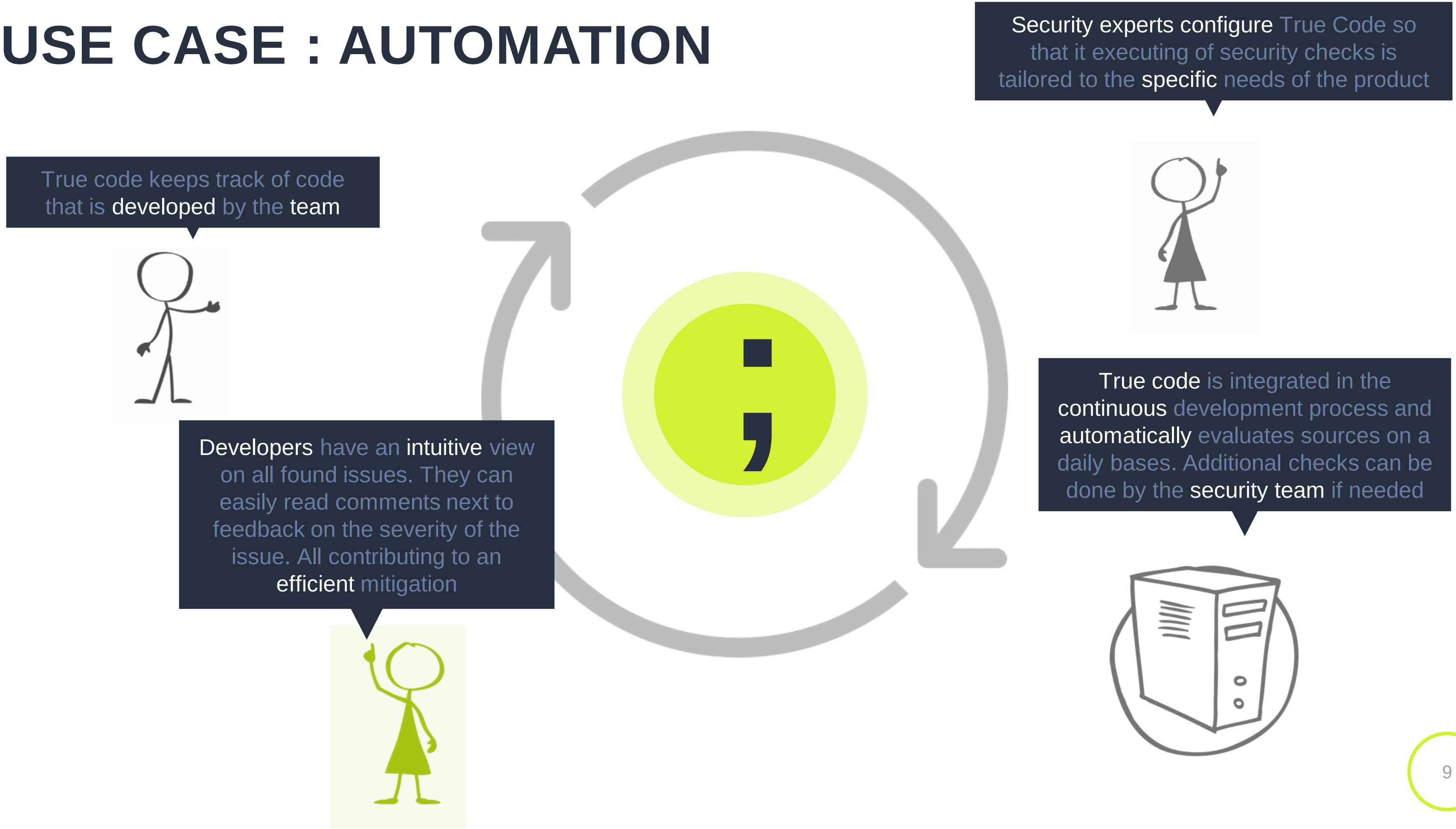


Security experts manually check the sources for security vulnerabilities. Found issues are marked and True Code keeps track of sources that have passed review

Developers have an intuitive view on all found issues. They can easily read comments next to feedback on the severity of the issue. All contributing to an efficient mitigation

Security experts can only check the mitigations, new sources and any code that has been changed after being reviewed

USE CASE : AUTOMATION



True code keeps track of code that is developed by the team

Security experts configure True Code so that it executing of security checks is tailored to the specific needs of the product

True code is integrated in the continuous development process and automatically evaluates sources on a daily bases. Additional checks can be done by the security team if needed

Developers have an intuitive view on all found issues. They can easily read comments next to feedback on the severity of the issue. All contributing to an efficient mitigation

Riscure B.V.

Frontier Building, Delftechpark 49
2628 XJ Delft
The Netherlands
Phone: +31 15 251 40 90
www.riscure.com

Riscure North America

550 Kearny St., Suite 330
San Francisco, CA 94108 USA
Phone: +1 650 646 99 79
inforequest@riscure.com

Riscure China

Room 2030-31, No. 989, Changle Road, Shanghai 200031
China
Phone: +86 21 5117 5435
inforcn@riscure.com

riscure

Challenge your security