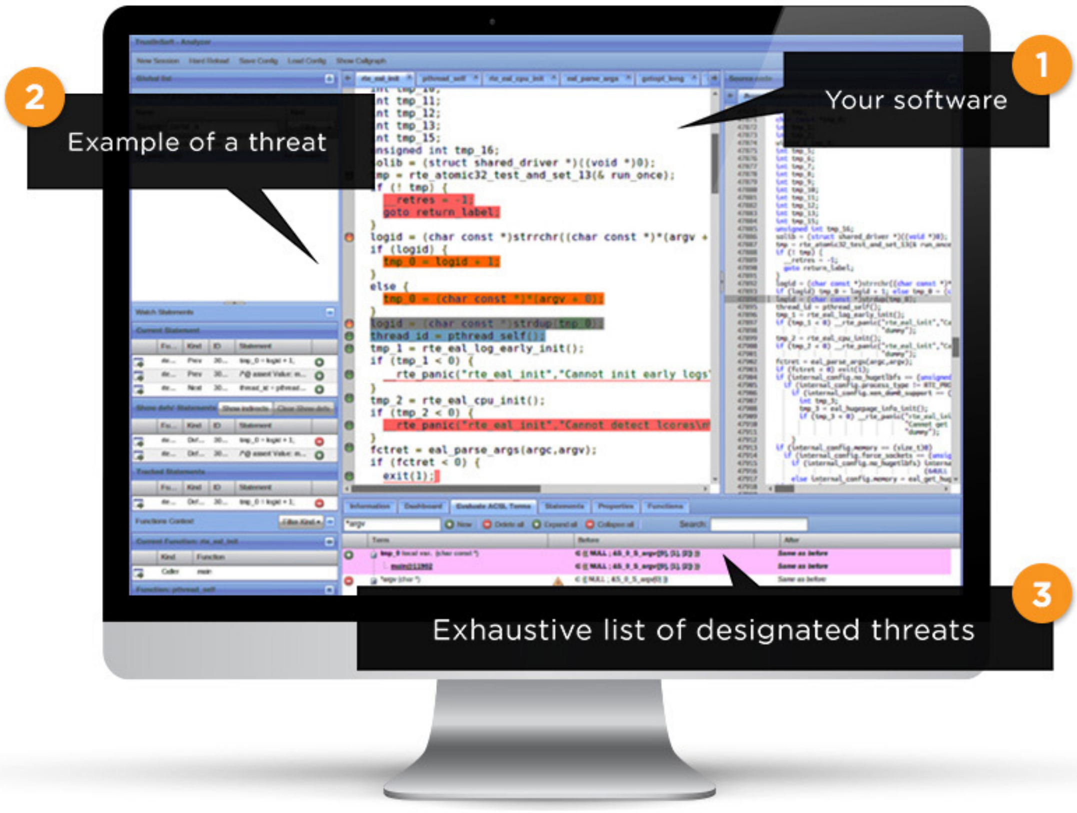
PROTECT CRITICAL INFRASTRUCTURE WITH TRUSTINSOFT

As cyber-threats loom large, protecting critical assets has never been more important or complex. Without robust cyber security policies, your data is at risk. That's where TrustInSoft comes in. TrustInSoft's source code analyzer is the best way to guarantee your software is free of a given flaw.

We're changing the rules of the game

The TrustInSoft Analyzer is an advanced static source code analyzer that performs comprehensive mathematical analyses of software to find and resolve source code flaws.

Our technology detects all vulnerabilities and is the only software that can mathematically guarantee that the most common security flaws are not present in your software.



```
nt tmp_11;
nt tmp_12;
nt tmp_13;
nt tmp_15;
nsigned int tmp_16;
olib = (struct shared_driver *)((void *)0);
mp = rte_atomic32_test_and_set_13(& run_once);
f (! tmp) {
  __retres = -1;
  goto return_label;
}
logid = (char const *)strrchr((char const *)*(argv +
if (logid) {
  tmp_0 = logid + 1;
}
else {
  tmp_0 = (char const *)*(argv + 0);
}
logid = (char const *)strdup(tmp_0);
thread_id = pthread_self();
tmp_1 = rte_eal_log_early_init();
if (tmp_1 < 0) {
  __rte_panic("rte_eal_init","Cannot init early logs'
}
tmp_2 = rte_eal_cpu_init();
if (tmp_2 < 0) {
  __rte_panic("rte_eal_init","Cannot detect lcores\n
}
fctret = eal_parse_args(argc,argv);
if (fctret < 0) {
  exit(1);
```

TRUST IN SOFT

Main Product Features

- Utilizes formal methods to provide mathematical guarantees
- Static analyzer framework with different built-in formal methods
- Web-based GUI with powerful built-in user-friendly tools
- Command-line interface for batch analyses and scripting
- Exhaustive debugger with access to all values of any variable

Vulnerabilities That Can Be Detected

VALID STRING:
Like Logic memory access, this alarm is emitted when using library internal built-ins such as strlen

POINTER COMPARISON:
Pointer comparisons whose results may vary from one compilation to another, such as &a < &b or &x+2 != NULL.

DIFFERING BLOCKS:
Subtractions between two pointers that may not point to the same allocated block.

NOT SEPARATED:
For example: y = x + x++; is undefined.

OVERLAP:
An assignment from lvalue to lvalue, the left and right lvalues must overlap either exactly or not at all.

DANGLING POINTER:
For example, returning the address of a local variable and use it outside its scope. Function type matches: Function pointer and pointed function have incompatible types.

USING TRUSTINSOFT ANALYZER IN THE SOFTWARE DESIGN PROCESS

TrustInSoft Analyzer is used by verification and validation teams to detect undefined behaviors, expand coverage, guarantee the absence of zero-day exploits, and ensure functional or security properties.

LEARN MORE AT **Trust-In-Soft.com**

TRUST IN SOFT