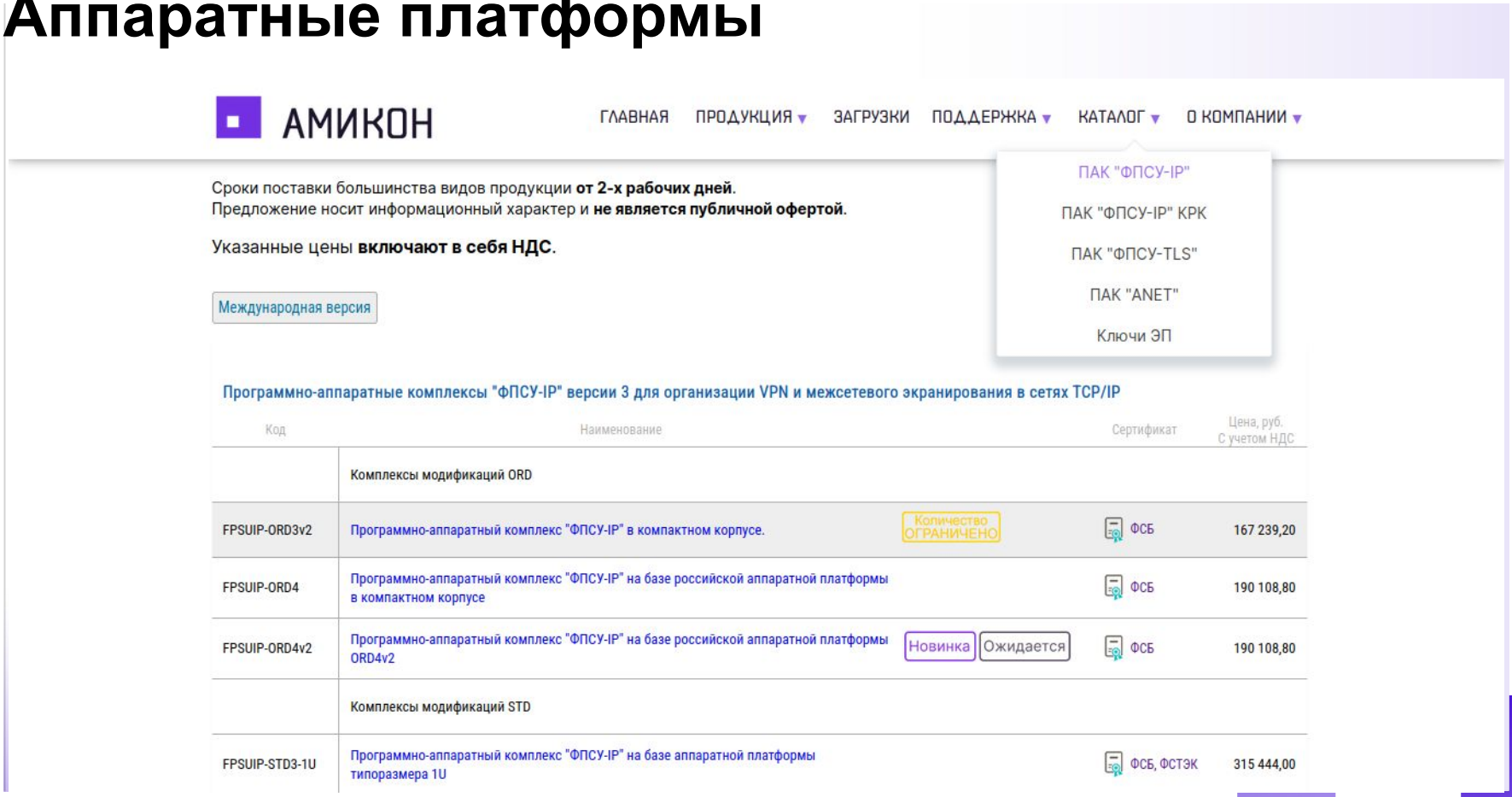
Безопасная разработка - наш опыт интеграции инструментов и практик

амикон

Аппаратные платформы

амикон

Аппаратные платформы



АМИКОН

ГЛАВНАЯ ПРОДУКЦИЯ ЗАГРУЗКИ ПОДДЕРЖКА КАТАЛОГ О КОМПАНИИ

ПАК "ФПСУ-IP"
ПАК "ФПСУ-IP" КРК
ПАК "ФПСУ-TLS"
ПАК "ANET"
Ключи ЭП

Сроки поставки большинства видов продукции **от 2-х рабочих дней**.
Предложение носит информационный характер и **не является публичной офертой**.

Указанные цены **включают в себя НДС**.

Международная версия

Программно-аппаратные комплексы "ФПСУ-IP" версии 3 для организации VPN и межсетевого экранирования в сетях TCP/IP

Код	Наименование		Сертификат	Цена, руб. С учетом НДС
	Комплексы модификаций ORD			
FPSUIP-ORD3v2	Программно-аппаратный комплекс "ФПСУ-IP" в компактном корпусе.	Количество ОГРАНИЧЕНО	ФСБ	167 239,20
FPSUIP-ORD4	Программно-аппаратный комплекс "ФПСУ-IP" на базе российской аппаратной платформы в компактном корпусе		ФСБ	190 108,80
FPSUIP-ORD4v2	Программно-аппаратный комплекс "ФПСУ-IP" на базе российской аппаратной платформы ORD4v2	Новинка Ожидается	ФСБ	190 108,80
	Комплексы модификаций STD			
FPSUIP-STD3-1U	Программно-аппаратный комплекс "ФПСУ-IP" на базе аппаратной платформы типоразмера 1U		ФСБ, ФСТЭК	315 444,00

С чего начинался наш анализ

Flawfinder is a simple program that scans C/C++ source code and reports potential security flaws. It can be a useful tool for examining software for vulnerabilities, and it can also serve as a simple introduction to static source code analysis tools more generally. It is designed to be easy to install and use. Flawfinder supports the Common Weakness Enumeration (CWE) and is officially CWE-Compatible.

Cppcheck is a static analysis tool for C/C++ code. It provides unique code analysis to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs. The goal is to have very few false positives. Cppcheck is designed to be able to analyze your C/C++ code even if it has non-standard syntax (common in embedded projects).

```
$ intercept-build make
$ cppcheck --project=compile_commands.json --enable=all --xml-version=2 \
  --xml -q 2> report.xml
$ cppcheck-htmlreport --file=report.xml --report-dir=out
```

Безопасная разработка - наши дни

- CI: Gitea + Jenkins
- Проверяем pull request'ы глазами и не только (Jenkins Gitea plugin)
- Clang Static Analyzer - от scan-build к CodeChecker
  - scan-build - Does not include support for cross-translation-unit analysis.
  - CodeChecker - CTU + Z3: --ctu --z3-refutation=on
- GCC warnings - Jenkins парсит лог; поддерживается новой версией CodeChecker'a
- Svace + Svacer
- Dependency-Track - отслеживаем уязвимости в зависимостях

Истории об интеграции средств статического анализа

#1: Clang SA с CTU-анализом под Windows

Удаленный администратор - собирается компилятором bcc32c (RAD Studio).
BCC32C is a Clang-enhanced compiler with a command line flag compatible only with BCC32 (the classic compiler).

Напрямую анализу не подлежит из-за расширений синтаксиса C++, которые добавлены в компилятор (и используются в подключаемых хедерах).

Clang-enhanced => доступ к бэкенду можно получить через флаг `-Xclang`.
Воспроизводим поведение scan-build/CodeChecker.

Получаем compilation database - файл compile_commands.json, внутри массив json-объектов вида:

```
{
  "arguments": [
    "bcc32c", "-DRELEASE",
    "-I", "include",
    "-isystem", "c:\\borland\\include",
    "-tM",
    "-Xdriver", "-emit-ast",
    "-o",
    "C:\\jenkins\\workspace\\clangsa\\ast\\source.ast",
    "C:\\jenkins\\workspace\\clangsa\\source.cpp"
  ],
  "file": "source.cpp",
  "directory": "C:\\jenkins\\workspace\\clangsa"
}
```

Генерируем AST-файлы и compile_commands.json:

```
arguments = ['bcc32c', *macros_args, *include_args, *borland_args, '-Xdriver', '-emit-ast']
compile_objects = run_compiler(sources, out_dir, 'ast', arguments)
```

Получаем файл externalDefMap.txt:

```
res = ''
for count,file in enumerate(sources):
  proc_args = ['clang-extdef-mapping', '-p', db_dir, file]
  res += subprocess.run(proc_args, capture_output=True, text=True).stdout
```

`clang-extdef-mapping`:

This tool collects the USR ('Unified Symbol Resolution' values are strings that provide an unambiguous reference to a symbol) name and location of external definitions in the source files (excluding headers).
Input can be either source files that are compiled with compile database or .ast files that are created from clang's -emit-ast option.

```
23:c:@S@CValue@F@SetClass# ast\DlgCfg.ast
15:c:@F@InitItems# ast\DlgCfg.ast
54:c:@S@TDiffCfgDsrDlg@F@FindValueinAllStruct#I#I#I#*v#I#
ast\DlgCfg.ast
37:c:@S@CStructure@F@AddItem#*$@S@CItem# ast\DlgCfg.ast
12:c:@F@GetPort ast\PortNames.ast
```

Запускаем анализ:

```
analyze_args = """
-Xdriver --analyze
-Xclang -analyzer-config
-Xclang experimental-enable-naive-ctu-analysis=true
-Xclang -analyzer-config
-Xclang ctu-dir={}
-Xclang -analyzer-config
-Xclang ctu-index-name={}
-Xclang -analyzer-config
-Xclang crosscheck-with-z3=true
-Xclang -analyzer-output=plist-multi-file
""".format(project_dir, extdef_file).split()
# extdef_file - file with TU external definitions

arguments = ['bcc32c', *macros_args, *include_args, *borland_args, *analyze_args]
compile_objects = run_compiler(sources, out_dir, 'plist', arguments)
```

После анализа получаем папку с результатами анализа в plist-файлах.
Конвертируем их в html с помощью CodeChecker:

```
$ codechecker parse -e html -o reports_html reports
```

Checker statistics

Checker name	Severity	Number of reports
core.CallAndMessage	H	10
core.NullDereference	H	30
core.UndefinedBinaryOperatorResult	H	10
core.uninitialized.Assign	H	2
core.uninitialized.Branch	H	2
cplusplus.NewDeleteLeaks	H	1
deadcode.DeadStores	L	89
unix.Malloc	M	1
unix.MallocSizeof	M	10
unix.cstring.NullArg	M	1

Severity statistics

Severity	Number of reports
H	55
M	12
L	89

#2: Clang SA false-positives

1. Разметка комментариями

```
void test_simple() {
  // codechecker_confirmed [clang-diagnostic-division-by-zero, core.DivideZero] Это реальный баг.
  // codechecker_intentional [clang-diagnostic-unused-variable] Это не ошибка.
  int x = 1 / 0;
}
```

2. Анализ с SMT-решателем Z3

Clang не поставляется с поддержкой Z3 (из-за лицензии на последний), в том числе в официальном apt.llvm.org. Сборка ложится на плечи тех, кому это нужно:

```
cmake -DLLVM_ENABLE_PROJECTS=clang -DLLVM_ENABLE_Z3_SOLVER=ON #...
```

Используем CodeChecker с флагом `--z3-refutation=on`.

Анализ на сутки

Профили CodeChecker'a: default, sensitive, extreme; portability.
Посмотреть список проверок для конкретного профиля:

```
$ CodeChecker checkers --analyzers=clangsa --profile=default \
   -o rows
```

Количество проверок для разных профилей (clang 15.0.7):

portability: 1 - optin.portability.UnixAPI
default: 45
sensitive: 72
extreme: 109

```
$ CodeChecker analyze -j64 -o reports_security \
  --analyzers clangsa --ctu --z3-refutation=on \
  -e profile:extreme compile_commands.json

[INFO 2024-01-17 22:22] - [541/546] clangsa analyzed 541.c successfully.
[INFO 2024-01-18 11:51] - [542/546] clangsa analyzed 542.c successfully.
[INFO 2024-01-18 17:36] - [543/546] clangsa analyzed 543.c successfully.
[INFO 2024-01-18 18:15] - [544/546] clangsa analyzed 544.c successfully.
[INFO 2024-01-18 21:00] - [545/546] clangsa analyzed 545.c successfully.
...
[INFO 2024-01-18 21:00] - Analysis length: 81703.43273234367 sec.
```

22 h 41 min 43 sec

#3: Деманглинг C++ имен в отчетах Svace(r)

Позиция: /.build/asyn/fpsutls/next/timeheap.cpp:53	Undecided
Функция	_ZN9THeapView6updateEv
Сообщение об ошибке	A string is copied into the buffer '&this->heapStr[0]' of size 32 without checking its length first at timeheap.cpp:53.

```
48
49    if ((curtick-last_tick)<=18) return; // ►► ►►►► 1 □► ► □
50    last_tick=curtick;
51    if ((newMem=GetFreeram())!=oldMem) {
52      oldMem=newMem;
53      if (oldMem<1024) strcpy(heapStr,"     ►►►....");
54      else sprintf(heapStr,"%11lu",newMem);
55      drawView();
56    }
57  }
58
```

амикон

Переписываем файл svres, пропуская манглированные имена через c++filt:

```
def demangle_func(m):
  mangled = m[1]
  if mangled == '--': return m[0]
  demangled = run(['c++filt', mangled], capture_output=True, text=True).stdout
  name = demangled.strip().replace('&', '&amp;')
  return f'function="{name}"

regex = [
  (re.compile('function="(_Z[^"]*)"', re.M), demangle_func),
  (re.compile('struct (_Z[^ ]*)'), demangle_struct)
]

for p, demangle in regex:
  content = p.sub(demangle, orig_content)
content = run(['c++filt'], input=content, capture_output=True, text=True).stdout
print(content, end='')
```

```
$ svace-demangle -h
usage: svace-demangle [-h] (-f FILE | -d DIR) [-i] [-b]

options:
  -h, --help          show this help message and exit
  -f FILE, --file FILE  Svace results file
  -d DIR, --dir DIR   directory with .svace-dir in it
  -i, --inplace       change results file in-place
  -b, --backup        store copy of results file with .bu extension

$ svace-demangle -d . -ib
demangling .svace-dir/shared/data/85/8586674ceba31235a9e1d360c68d9177ed406465
demangling finished.
copy of original results file:
.svace-dir/shared/data/85/8586674ceba31235a9e1d360c68d9177ed406465.bu
```

```
$ svacer import ...
$ svacer upload ...
```

Позиция: /.build/asyn/fpsutls/text/timeheap.cpp:53	False positive
Функция	THeapView::update()
Сообщение об ошибке	A string is copied into the buffer '&this->heapStr[0]' of size 32 without checking its length first at timeheap.cpp:53.

```
48
49    if ((curtick-last_tick)<=18) return; // ►► ►►►► 1 □► ► □
50    last_tick=curtick;
51    if ((newMem=GetFreeram())!=oldMem) {
52      oldMem=newMem;
53      if (oldMem<1024) strcpy(heapStr,"     ►►►....");
54      else sprintf(heapStr,"%11lu",newMem);
55      drawView();
56    }
57  }
58
```

амикон

Фаззинг

- Обучение (курсы / обучение сотрудников / research)
- Какие инструменты попробовали: libFuzzer, AFL++, FUTAG
- Инструменты на этапе внедрения: Natch, Crusher (Sydr)
- Интересные направления:
    - Hopper - аналог syzkaller для C-библиотек, automatic intra- and inter-API constraints learning.
    - oss-fuzz-gen - generates fuzz targets for real-world C/C++ projects with various Large Language Models (LLM) and benchmarks them via the OSS-Fuzz
    - DeepState - unit test-like interface for fuzzing and symbolic execution

Спасибо за внимание!

амикон