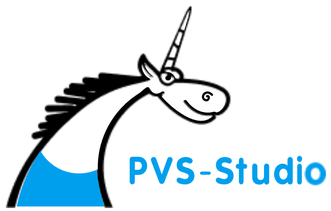

PVS-Studio

Процессы разработки безопасного ПО



Андрей Карпов

ООО «ПВС», CBDO

Андрей Карпов

Директор по развитию бизнеса (CBDO)

- Один из основателей проекта PVS-Studio – https://pvs-studio.ru/
- 17 лет в сфере качества и анализа кода
- Хабр: @Andrey2008

- Email: karpov (@) viva64.com

PVS-Studio

Что первое приходит в голову, когда говорят о качестве и безопасности кода?

К сожалению, скорее всего, тестирование

- Тестирование само по себе нужно и полезно
- Но сразу уже всё поставлено с ног на голову

- Мы пытаемся решить проблему, максимально ускоряя процесс проектирования, чтобы в конце работы над системой у нас осталось достаточно времени для нахождения ошибок, допущенных из-за слишком быстрого проектирования.
  Гленфорд Майерс (Glenford Myers)

Но качество обеспечивается, в первую очередь, процессами разработки

- Надёжность программы достигается, в первую очередь, благодаря её правильному проектированию, а не бесконечному тестированию.
  Юров В.И.

Хорошо, какие ещё процессы вспоминаются?

- Оформление кода (стандарты кодирования);
- Обучение сотрудников;
- Статический и динамический анализ кода;
- Функциональное тестирование;
- Прочее.

- Ещё можно воспользоваться поиском в Интернете ☺

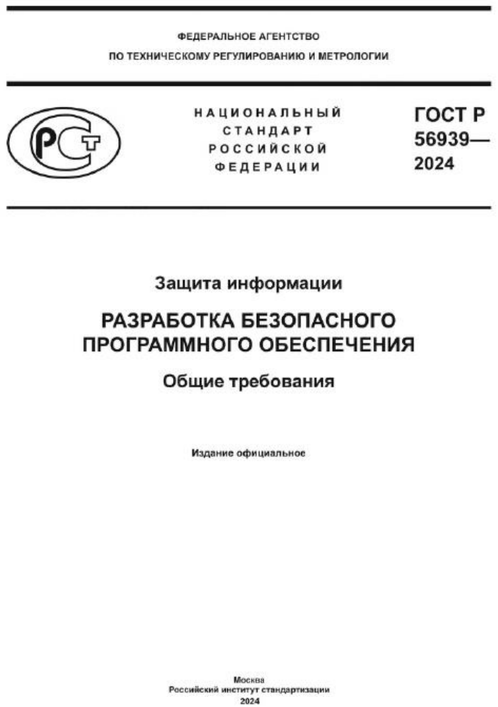
Лайфхак: вспоминать или искать не надо, а надо лишь…

- … посмотреть в уже созданные документы!
- Всё уже давно собрано и подготовлено
- Во-первых, есть
  AppSec Table Top: методология безопасной разработки от Positive Technologies

https://static.ptsecurity.com/docs/knowledge-base/metodologiya-appsec-table-top.pdf

Во-вторых, ГОСТ Р 56939-2024 – Разработка безопасного ПО

- Пожалуй, этот стандарт стоит смотреть в первую очередь:
  - он вышел совсем недавно и поэтому наиболее полно описывает современные практики/подходы к созданию безопасного ПО;
  - он имеет юридическую силу на территории РФ.



ФЕДЕРАЛЬНОЕ АГЕНТСТВО
ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р 56939— 2024

Защита информации

РАЗРАБОТКА БЕЗОПАСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Общие требования

Издание официальное

Москва
Российский институт стандартизации
2024

Список разработчиков внушителен. Среди них и Positive Technologies

Федеральной службой по техническому и экспортному контролю (**ФСТЭК** России), Акционерным обществом «**Лаборатория Касперского**» (АО «Лаборатория Касперского»), Федеральным государственным бюджетным учреждением науки Институт системного программирования им. В.П. Иванникова Российской академии наук (**ИСП РАН**), Акционерным обществом «Информационные технологии и коммуникационные системы» (АО «**ИнфоТеКС**»), Акционерным обществом «**ПозитивТекнолоджиз**» (АО «Позитив Текнолоджиз»), Обществом с ограниченной ответственностью «**РусБИ-Тех-Астра**» (ООО «РусБИТех-Астра»), Акционерным обществом «Сбербанк-Технологии» (АО «**Сбер-Тех**»), Обществом с ограниченной ответственностью Научно-технический центр «**Фобос-НТ**» (ООО НТЦ «Фобос-НТ»), Обществом с ограниченной ответственностью «Центр безопасности информации» (ООО «**ЦБИ**»), Акционерным обществом «Научно-производственное объединение «**Эшелон**» (АО НПО «Эшелон»).

ГОСТ Р 56939-2024 описывает 25 процессов для построения РБПО

- Можно брать и внедрять наиболее актуальное и полезное для ваших проектов
- Рассмотрим некоторые из описанных рекомендаций

- P.S. Поддерживать стандарт целиком нужно только в случае, если требуется сертификация ФСТЭК. Это уже отдельная история.

5.1. Планирование процессов разработки безопасного программного обеспечения

- Важный экзистенциальный пункт ☺
- 5.1.1.1. Обеспечение потребностей в **ресурсах**, необходимых для реализации процессов разработки безопасного ПО

- Не получится просто «Саша, Маша, с завтрашнего дня пишем безопасный код»

Понятно, что нужны ресурсы. И что?

- Как раз часто непонятно
- Это хороший язык взаимодействия между программистами, менеджерам и руководством
- Внедрение РБПО влечёт удорожание разработки на **10-15%**
- Если не заложено время и бюджет на внедрение даже самых хороших практик, ничего не получится
- Грамотно обосновывать бюджет – это тоже важный процесс РБПО! ☺

5.2. Обучение сотрудников. Цели:

- Получение информации о существующих практиках, обучающих курсах и тренингах по РБПО
- Организация обучения сотрудников
- Создание условий для снижения количества ошибок и уязвимостей

5.5. Управление недостатками и запросами на изменение ПО

- Цели:
  - управление недостатками ПО;
  - управление запросами на изменения.
- Суть: по каким правилам происходит работа с ошибками и запросами на доработки в системе управления задачами (багтрекере):
  - как описывать недостатки;
  - приоритет выполнения действий;
  - и т.д.

5.7. Моделирование угроз и разработка описания поверхности атаки

- Чтобы рационально использовать силы для анализа кодовой базы, нужно определить поверхность атаки
- Поверхность атаки — это потенциально уязвимые функции и модули, обрабатывающие пользовательский ввод или чувствительные данные, а также интерфейсы, через которые эти данные поступают
- Зачем искать поверхность атаки для своего проекта: habr.com/ru/companies/isp_ras/articles/801459/

Определение поверхности атаки

- Эксперт

- Статический анализ кода
- Правильнее говорить: **поиск проблем на поверхности атаки**, а не самой поверхности
  - taint-анализ (анализ помеченных данных) – pvs-studio.ru/ru/blog/terms/6496/
  - **taint-анализ малополезен без дополнительной разметки истоков и стоков**
  - любой статический анализатор, соответствующий ГОСТ Р 71207-2024, согласно п.7.4 должен выполнять межпроцедурный и межмодульный контекстно-чувствительный анализ помеченных данных
  - Пример инструмента: PVS-Studio – pvs-studio.ru/ru/pvs-studio/gost-71207/

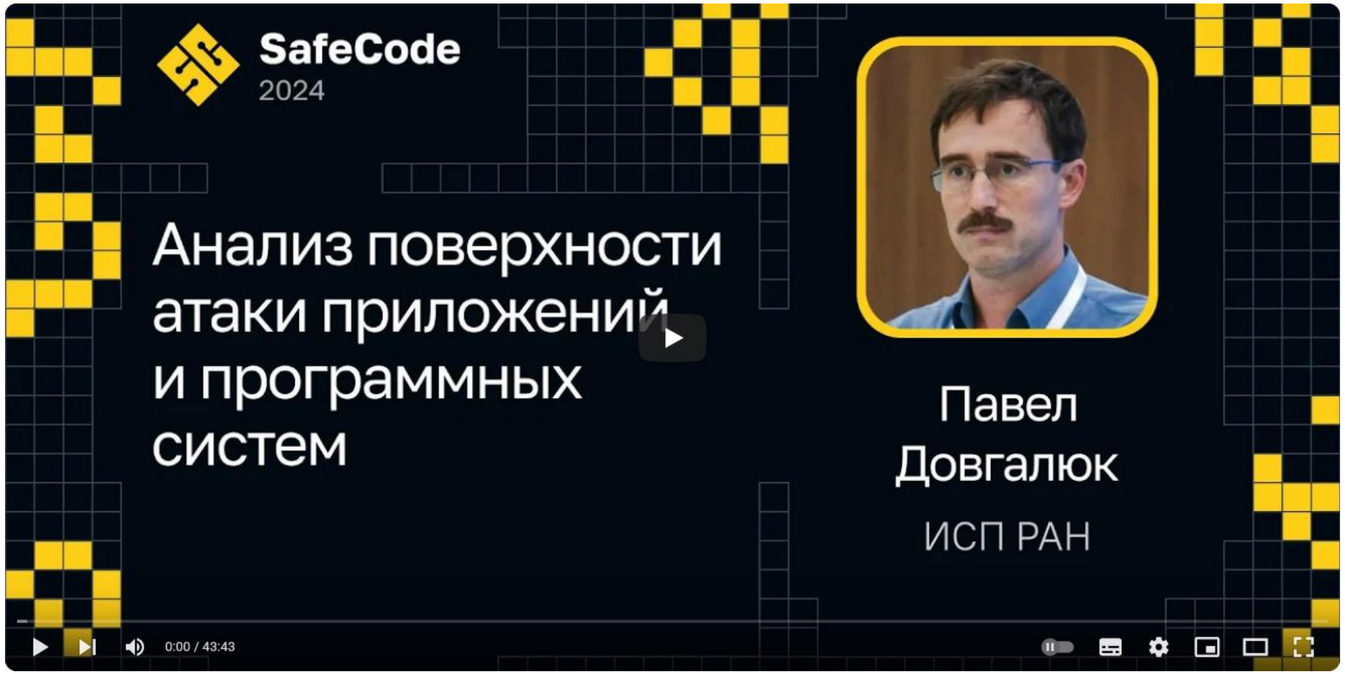
Определение поверхности атаки

- Поиск точек входа:
  - AttackSurfaceAnalyzer – github.com/microsoft/AttackSurfaceAnalyzer
  - Nikto – github.com/sullo/nikto

Определение поверхности атаки

- Динамический анализ кода
- Пример: Natch (ИСП РАН)
  - Страница продукта: www.ispras.ru/technologies/natch/
  - Исходный код: github.com/ispras/natch
  - Как найти поверхность атаки незнакомых приложений с помощью Natch: habr.com/ru/companies/isp_ras/articles/788490/
  - Разглядываем CodeScoring с помощью Natch: habr.com/ru/companies/isp_ras/articles/892548/

Дополнительный материал



SafeCode 2024. Павел Довгалюк — Анализ поверхности атаки приложений и программных систем.
Запись: youtu.be/wcsMnhb94YA?si=oFWIjtcEeNKAYJjw
Презентация: PDF

5.8. Формирование и поддержание в актуальном состоянии правил кодирования

- Единообразие оформления важно, так как большую часть времени разработчики не пишут, а читают код

- Учёт в регламенте (стандарте кодирования) использования опасных конструкций языка программирования

Реализация

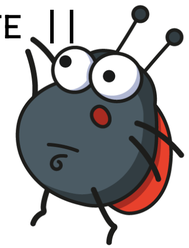
- Можно оттолкнуться от этой подборки Coding Standards: isocpp.org/wiki/faq/coding-standards

- Книга Стива Макконнелла «Совершенный код»

Примеры, что можно включить в стандарт

- TDengine, язык C
- Всматриваться не хочется? Да?

```
static bool rpcNoDelayMsg(tmsg_t msgType) {
  if (msgType == TDMT_VND_FETCH_TTL_EXPIRED_TBS || msgType == TDMT_VND_S3MIGRATE || msgType == TDMT_VND_S3MIGRATE ||
      msgType == TDMT_VND_QUERY_COMPACT_PROGRESS || msgType == TDMT_VND_DROP_TTL_TABLE) {
    return true;
  }
  return false;
}
```



«Код-колбаса» провоцирует ошибки


```
pcNoDelayMsg(tmsg_t msgType) {
 == TDMT_VND_FETCH_TTL_EXPIRED_TBS || msgType == TDMT_VND_S3MIGRATE || msgType == TDMT_VND_S3MIGRATE ||
 == TDMT_VND_QUERY_COMPACT_PROGRESS || msgType == TDMT_VND_DROP_TTL_TABLE) {
ue;

e;
```

PVS-Studio: V501 There are identical sub-expressions 'msgType == TDMT_VND_S3MIGRATE' to the left and to the right of the '||' operator. dmTransport.c 398

Форматирование кода таблицей


```
public static bool Equals(ref UInt256 a, ref UInt256 b)
{
  return a.s0 == b.s0 &&
         a.s1 == b.s1 &&
         a.s2 == b.s2 &&
         a.s2 == b.s2;
}
```

Ошибка лучше заметна

Ещё лучше: логические операторы слева


```
public static bool Equals(ref UInt256 a, ref UInt256 b)
{
  return     a.s0 == b.s0
          && a.s1 == b.s1
          && a.s2 == b.s2
          && a.s2 == b.s2;
}
```

Почему лучше, станет понятно
из следующего примера

Ошибка заметнее, но много пробелов


```
static bool rpcNoDelayMsg(tmsg_t msgType) {
  if (msgType == TDMT_VND_FETCH_TTL_EXPIRED_TBS ||
      msgType == TDMT_VND_S3MIGRATE             ||
      msgType == TDMT_VND_S3MIGRATE             ||
      msgType == TDMT_VND_QUERY_COMPACT_PROGRESS||
      msgType == TDMT_VND_DROP_TTL_TABLE) {     ||
    return true;
  }
  return false;
}
```

Всё переформатировать? ;(


```
static bool rpcNoDelayMsg(tmsg_t msgType) {
  if (msgType == TDMT_VND_FETCH_TTL_EXPIRED_TBS  ||
      msgType == TDMT_VND_S3MIGRATE              ||
      msgType == FIX_FIX_FOOOOOOOOOOOOOOOOOOOOOO ||
      msgType == TDMT_VND_QUERY_COMPACT_PROGRESS ||
      msgType == TDMT_VND_DROP_TTL_TABLE) {      ||
    return true;
  }
  return false;
}
```

Так лучше


```
static bool rpcNoDelayMsg(tmsg_t msgType) {
  if (   msgType == TDMT_VND_FETCH_TTL_EXPIRED_TBS
      || msgType == TDMT_VND_S3MIGRATE
      || msgType == TDMT_VND_S3MIGRATE
      || msgType == TDMT_VND_QUERY_COMPACT_PROGRESS
      || msgType == TDMT_VND_DROP_TTL_TABLE) {
    return true;
  }
  return false;
}
```

Код можно сделать ещё изящнее и короче


```
static bool rpcNoDelayMsg(tmsg_t msgType) {
  return     msgType == TDMT_VND_FETCH_TTL_EXPIRED_TBS
         || msgType == TDMT_VND_S3MIGRATE
         || msgType == TDMT_VND_QUERY_COMPACT_PROGRESS
         || msgType == TDMT_VND_DROP_TTL_TABLE);
}
```

Форматирование кода таблицей

- Подробнее: https://pvs-studio.ru/ru/blog/terms/7003/

Автоматическое форматирование

- Рекомендуется использовать программные средства автоматической проверки правил кодирования:
  - ClangFormat – clang.llvm.org/docs/ClangFormat.html
  - Uncrustify – github.com/uncrustify/uncrustify

5.10. Статический анализ исходного кода

- Разновидность автоматизированного обзора кода
- Сильные стороны:
  - Статические анализаторы проверяют даже те фрагменты кода, которые выполняются крайне редко;
  - Раннее обнаружение ошибок;
  - Возможно находить ошибки, которые невидимы для других инструментов, например, опечатки;
  - Возможность проверять юнит-тесты;
  - Возможность найти ошибочные паттерны, про которые не знают программисты.

Поиск ошибок-невидимок: опечатки


```
static int ot_ipsec_sa_common_param_fill(....)
{
  ....
  if (w2->s.enc_type == ROC_IE_OT_SA_ENC_AES_CBC ||
      w2->s.enc_type == ROC_IE_OT_SA_ENC_AES_CCM ||
      w2->s.enc_type == ROC_IE_OT_SA_ENC_AES_CTR ||
      w2->s.enc_type == ROC_IE_OT_SA_ENC_AES_GCM ||
      w2->s.enc_type == ROC_IE_OT_SA_ENC_AES_CCM ||
      w2->s.auth_type == ROC_IE_OT_SA_AUTH_AES_GMAC) {
```

Непонятно, на что ругаться динамическом анализатору.

Юнит-тестом найти такое можно, но сложно.

В случае статического анализа всё просто


```
static int ot_ipsec_sa_common_param_fill(....)
{
  ....
  if (w2->s.enc_type == ROC_IE_OT_SA_ENC_AES_CBC ||
      w2->s.enc_type == ROC_IE_OT_SA_ENC_AES_CCM ||
      w2->s.enc_type == ROC_IE_OT_SA_ENC_AES_CTR ||
      w2->s.enc_type == ROC_IE_OT_SA_ENC_AES_GCM ||
      w2->s.enc_type == ROC_IE_OT_SA_ENC_AES_CCM ||
      w2->s.auth_type == ROC_IE_OT_SA_AUTH_AES_GMAC) {
```

Проект DPDK, C. PVS-Studio: V501 There are identical sub-expressions 'w2->s.enc_type == ROC_IE_OT_SA_ENC_AES_CCM' to the left and to the right of the '||' operator. cnxk_security.c 177

Проверка юнит-тестов


```
TEST_ASSERT(dsqp.service_cleanup_delay.sec = 4);
TEST_ASSERT(dsqp.service_cleanup_delay.nanosec = 2000);
TEST_ASSERT(dsqp.history_kind == KEEP_LAST_HISTORY_QOS);
TEST_ASSERT(dsqp.history_depth == 172);
TEST_ASSERT(dsqp.max_samples == 389);
```

Data Distribution Service, C++. PVS-Studio:

- V559 Suspicious assignment inside the condition expression of 'if' operator: dsqp.service_cleanup_delay.sec = 4. ut_parameterlistconverter.cpp 1295
- V559 Suspicious assignment inside the condition expression of 'if' operator. ut_parameterlistconverter.cpp 1296


ты должен был бороться со злом
а не примкнуть к нему

Поиск незнакомых паттернов ошибок


```
char c;
printf("%s is already in *.base_fs format, just
copying ....);
rewind(blk_alloc_file);
while ((c = fgetc(blk_alloc_file)) != EOF) {
  fputc(c, base_fs_file);
```

Кто видит ошибку?

Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ
0	0	спец. NOP	32	20	спец. SP (Пробел)	64	40	@	96	60	`	128	80	Ъ	160	A0		192	C0	А	224	E0	а
1	1	спец. SOH	33	21	!	65	41	A	97	61	a	129	81	Г	161	A1	Ў	193	C1	Б	225	E1	б
2	2	спец. STX	34	22	"	66	42	B	98	62	b	130	82	‚	162	A2	ў	194	C2	В	226	E2	в
3	3	спец. ETX	35	23	#	67	43	C	99	63	c	131	83	ѓ	163	A3	J	195	C3	Г	227	E3	г
4	4	спец. EOT	36	24	$	68	44	D	100	64	d	132	84	„	164	A4	¤	196	C4	Д	228	E4	д
5	5	спец. ENQ	37	25	%	69	45	E	101	65	e	133	85	…	165	A5	Ґ	197	C5	Е	229	E5	е
6	6	спец. ACK	38	26	&	70	46	F	102	66	f	134	86	†	166	A6	¦	198	C6	Ж	230	E6	ж
7	7	спец. BEL	39	27	'	71	47	G	103	67	g	135	87	‡	167	A7	§	199	C7	З	231	E7	з
8	8	спец. BS	40	28	(	72	48	H	104	68	h	136	88	€	168	A8	Ё	200	C8	И	232	E8	и
9	9	спец. Табуляция	41	29	)	73	49	I	105	69	i	137	89	‰	169	A9	©	201	C9	Й	233	E9	й
10	0A	спец. LF (Возвр. каретки)	42	2A	*	74	4A	J	106	6A	j	138	8A	Љ	170	AA	Є	202	CA	К	234	EA	к
11	0B	спец. VT	43	2B	+	75	4B	K	107	6B	k	139	8B	‹	171	AB	«	203	CB	Л	235	EB	л
12	0C	спец. FF	44	2C	,	76	4C	L	108	6C	l	140	8C	Њ	172	AC	¬	204	CC	М	236	EC	м
13	0D	спец. CR (Новая строка)	45	2D	-	77	4D	M	109	6D	m	141	8D	Ќ	173	AD	-	205	CD	Н	237	ED	н
14	0E	спец. SO	46	2E	.	78	4E	N	110	6E	n	142	8E	Ћ	174	AE	®	206	CE	О	238	EE	о
15	0F	спец. SI	47	2F	/	79	4F	O	111	6F	o	143	8F	Џ	175	AF	Ї	207	CF	П	239	EF	п
16	10	спец. DLE	48	30	0	80	50	P	112	70	p	144	90	ђ	176	B0	°	208	D0	Р	240	F0	р
17	11	спец. DC1	49	31	1	81	51	Q	113	71	q	145	91	‘	177	B1	±	209	D1	С	241	F1	с
18	12	спец. DC2	50	32	2	82	52	R	114	72	r	146	92	’	178	B2	I	210	D2	Т	242	F2	т
19	13	спец. DC3	51	33	3	83	53	S	115	73	s	147	93	“	179	B3	i	211	D3	У	243	F3	у
20	14	спец. DC4	52	34	4	84	54	T	116	74	t	148	94	”	180	B4	г	212	D4	Ф	244	F4	ф
21	15	спец. NAK	53	35	5	85	55	U	117	75	u	149	95	•	181	B5	µ	213	D5	Х	245	F5	х
22	16	спец. SYN	54	36	6	86	56	V	118	76	v	150	96	–	182	B6	¶	214	D6	Ц	246	F6	ц
23	17	спец. ETB	55	37	7	87	57	W	119	77	w	151	97	—	183	B7	·	215	D7	Ч	247	F7	ч
24	18	спец. CAN	56	38	8	88	58	X	120	78	x	152	98	[illegible]	184	B8	ё	216	D8	Ш	248	F8	ш
25	19	спец. EM	57	39	9	89	59	Y	121	79	y	153	99	™	185	B9	№	217	D9	Щ	249	F9	щ
26	1A	спец. SUB	58	3A	:	90	5A	Z	122	7A	z	154	9A	љ	186	BA	є	218	DA	Ъ	250	FA	ъ
27	1B	спец. ESC	59	3B	;	91	5B	[	123	7B	{	155	9B	›	187	BB	»	219	DB	Ы	251	FB	ы
28	1C	спец. FS	60	3C	<	92	5C	\	124	7C	|	156	9C	њ	188	BC	j	220	DC	Ь	252	FC	ь
29	1D	спец. GS	61	3D	=	93	5D	]	125	7D	}	157	9D	ќ	189	BD	S	221	DD	Э	253	FD	э
30	1E	спец. RS	62	3E	>	94	5E	^	126	7E	~	158	9E	ћ	190	BE	s	222	DE	Ю	254	FE	ю
31	1F	спец. US	63	3F	?	95	5F	_	127	7F		159	9F	џ	191	BF	ї	223	DF	Я	255	FF	я

Windows кодировка CP1251.
Буква 'я' имеет код 255.

Проект Android, C: 'я' как EOF


```
char c;
printf("%s is already in *.base_fs format, just
copying ....);
rewind(blk_alloc_file);
while ((c = fgetc(blk_alloc_file)) != EOF) {
  fputc(c, base_fs_file);
```

PVS-Studio: V739 CWE-20 EOF should not be compared with a value of the 'char' type. The '(c = fgetc(blk_alloc_file))' should be of the 'int' type. blk_alloc_to_base_fs.c 61

Статический анализ исходного кода

- Слабые стороны:
  - Слаб в выявлении некоторых классов ошибок (утечки памяти, ошибки синхронизации и т.д.);
  - Большое количество ложно-положительных срабатываний, которые вытекают из самой методологии статического анализа кода.

- Впрочем, не надо выбирать. Надо использовать сразу несколько подходов.

Если захочется глубоко заглянуть, то

- Обзора PVS-Studio с точки зрения нового стандарта ГОСТ Р 71207-2024 (Статический анализ кода)
  pvs-studio.ru/ru/gost-manual/

Что там? Например:

- С точки зрения РБПО, в первую очередь важно выявлять баги, с наибольшей вероятностью влияющие на безопасность кода
- Виды дефектов перечислены в ГОСТ Р 71207–2024 (п. 6.3.-6.5.) и называются «**критическими ошибками**» (п. 3.1.13.):
  - ошибки непроверенного использования чувствительных данных;
  - ошибки переполнения буфера;
  - ошибки разыменования нулевого указателя;
  - и т.д.

Всё про ГОСТ Р 71207–2024

- Вебинары про ГОСТ Р 71207–2024:
  - Общее описание и актуальность – pvs-studio.ru/ru/blog/video/11050/
  - Терминология – pvs-studio.ru/ru/blog/video/11066/
  - Критические ошибки – pvs-studio.ru/ru/blog/video/11084/
  - Технологии анализа кода – pvs-studio.ru/ru/blog/video/11092/
  - Процессы – pvs-studio.ru/ru/blog/video/11102/
- ФСТЭК объявила о начале испытаний анализаторов: ib-bank.ru/bisjournal/news/21667

  Что это значит?

5.11. Динамический анализ кода программы

- Под динамическим анализом понимается в том числе и фаззинг-тестирование
- Инструментов много:
  - Valgrind – valgrind.org/
  - AddressSanitizer – clang.llvm.org/docs/AddressSanitizer.html
  - ThreadSanitizer – clang.llvm.org/docs/ThreadSanitizer.html
  - MemorySanitizer – clang.llvm.org/docs/MemorySanitizer.html
  - ИСП Fuzzer – www.ispras.ru/technologies/crusher/
  - и т.д.
- Но стоит учитывать, что на подходе соответствующий ГОСТ.

5.14. Управление доступом и контроль целостности кода при разработке ПО

- Регламент, где что хранится, какие права доступа имеют различные сотрудники (дополнительная защита от закладок и других неприятностей):
  - **принцип минимизации привилегий и разделения полномочий**;
  - обязанности сотрудников, их права и роли при разработке ПО;
  - правила хранения исходного кода ПО, включая правила резервного копирования исходного кода;
  - правила внесения изменений (модификации, добавления, удаления) в исходный код ПО.

5.15. Обеспечение безопасности используемых секретов

- Под секретами понимаются:
  - данные, которые могут использоваться для обеспечения аутентификации, целостности;
  - конфиденциальная информация (пароли, цифровые сертификаты и т. п.);
  - и т.д.
- Требования применяются пользователями стандарта по их усмотрению и в необходимых им объемах.

Регламент использования секретов может содержать:

- обязанности сотрудников и их роли при использовании секретов;
- зоны ответственности подразделений и сотрудников;
- порядок предоставления доступа к секретам;
- типы секретов, сроки их эксплуатации, действия при их компрометации;
- порядок формирования и хранения секретов;
- порядок ротации секретов;
- требования к системам хранения секретов;
- порядок подписи исполняемого кода ПО;
- и т.д.

5.16. Использование инструментов композиционного анализа

- Автоматизированное сканирование ПО с целью нахождения фрагментов с открытым исходным кодом и их дальнейшей проверки. Позволяет выявить возможные уязвимости в сторонних компонентах, устаревшие элементы, а также проблемы с лицензированием.
- CodeScoring – codescoring.ru
- Solar appScreener – rt-solar.ru/products/solar_appscreener/
- И другие – github.com/magnologan/awesome-sca

5.18. Функциональное тестирование

- Функциональное тестирование проводится для оценки соответствия ПО заданным функциональным требованиям
- Включает разные виды тестирования, позволяющие убедиться, что «**ПО делает то, что должно делать**»

- Функциональное тестирование проводится по принципу черного ящика, в связи с чем функциональность ПО можно протестировать, не зная принципа его внутренней работы

Типы функционального тестирования

- Ручное тестирование;
- Автоматизированное тестирование;
- Дымовой тест;
- Регрессионное тестирование;
- Юзабилити-тестирование;
- Интеграционные тесты;
- Тестирование производительности;
- И т.д.

5.19. Нефункциональное тестирование

- Функциональное тестирование проверяет, что «**ПО делает то, что должно делать**»

- Нефункциональное тестирование проверяет:
  1. **Как работает приложение** (файлы логирования очень большие, что может вызвать быстрое исчерпание дискового пространства)
  2. **ПО не делает то, что не должно делать**

При нефункциональном тестировании могут выполняться следующие проверки:

- сетевых взаимодействий ПО;
- локальных интерфейсов взаимодействия ПО;
- производительности функционирования ПО;
- операций, выполняемых с высокими привилегиями;
- работы с конфиденциальными данными;
- корректности выполнения файловых операций;
- реализации защищённости бинарных файлов;
- реализации системы управления секретами;
- возможности нарушения логики работы программы (например, DoS-атаки уровня приложения);
- безопасности реализации механизмов аутентификации и авторизации;
- безопасности обработки данных, полученных от потенциального нарушителя;
- и т.д.

5.23. Реагирование на информацию об уязвимостях

- Может, слышали такие истории, что кто-то отписал разработчикам про найденную уязвимость, а в ответ игнор и тишина?
- Чтобы такого не было, и нужна разработка регламента реагирования на информацию об уязвимостях

Некоторые артефакты

- обязанности сотрудников и их роли при реагировании на информацию об уязвимостях;
- правила реагирования на информацию об уязвимостях;
- правила оценки актуальности и критичности уязвимости;
- шаблон ответа на запросы пользователей об ошибках (уязвимостях) ПО (о применимости информации о найденных уязвимостях).

5.24. Поиск уязвимостей в программном обеспечении при эксплуатации

- Если выпущен релиз, не значит, что можно расслабиться. Скорее наоборот. Что ещё можно сделать?
- Регулярно искать в открытых источниках информацию об уязвимостях вашего ПО или в его сторонних компонентах
- Проводить более глубокий анализ инструментальными средствами
- **Поиск уязвимостей за вознаграждение (багбаунти)**

Если вдруг...
сертификация

Ориентация на процесс

- Компаниям, потенциально рассматривающим в будущем сертификацию по ГОСТ Р 56939-2024, полезно заранее более внимательно подойти к выбору инструментов и их регулярному использованию
- В первую очередь оценивается процесс разработки
- **Становится неуместной практика приобретения лицензий на несколько месяцев на этапе прохождения сертификаций**
- Помним, что на подходе новые ГОСТы

Чем стоит руководствоваться при выборе инструментов?

- Из нашего опыта взаимодействия с сертификационными лабораториями при выборе подходящего инструмента стоит обратить внимание на следующие параметры:
  - Инструмент должен быть включён в Реестр российского ПО **или** быть открытым (open source)
  - Соответствует требованиям "Методики выявления уязвимостей и недекларированных возможностей в программном обеспечении" от 25 декабря 2020 г.
  - Если говорить про статический анализ, то учитывает требования ГОСТ Р 71207-2024 (Статический анализ кода)

PVS-Studio может быть использован для сертификации

- Включён в Реестр российского ПО (запись № 9837)
- **Соответствует требованиям «Методики выявления уязвимостей и НДВ в ПО»**
- Разрабатывается с учётом требования ГОСТ Р 71207-2024
- На практике используется сертификационными лабораториями. За подробностями предлагаем обращаться в поддержку - Форма обратной связи / +7(903)844-02-22

Выводы

Учитесь лучшим практикам и внедряйте их

- Серьёзным программистам я хочу сказать: **уделяйте часть рабочего дня изучению и улучшению собственных методик**.
  Хотя на программистов всегда давит какой-то будущий или прошедший срок, методологическая абстракция – мудрая долговременная инвестиция.

  Роберт У. Флойд (Robert W. Floyd)

Общая рекомендация

- Пишите простой и красивый код: его будет легче понять, а ошибки в нём будут заметнее.

Я (Андрей Карпов)

Интересное для чтения

Вредные советы для C++ программистов

- Электронный вариант
  https://pvs-studio.ru/ru/book-giveaway/

Подписывайтесь на цикл публикаций по теме РБПО

- Бестиарий программирования
  https://t.me/programming_tales

Ответы на вопросы

ООО «ПВС»

Сайт: pvs-studio.ru

Андрей Карпов

Директор по развитию бизнеса

Презентацию подготовил

- Карпов Андрей Николаевич, 1981.
- ООО «ПВС», Директор по развитию бизнеса.
- Более 17 лет занимается темой статического анализа кода и качества программного обеспечения. Автор большого количества статей, посвящённых написанию качественного кода на языке C++. Один из основателей проекта PVS-Studio. Долгое время являлся CTO компании и занимался разработкой C++ ядра анализатора. Основная деятельность на данный момент — развитие компании, обучение сотрудников и DevRel деятельность.
- Другая информация и контакты.