セガ社も認めた静的解析

ゲーム開発からバグを取り除く方法

株式会社セガ　節政 暁生
株式会社セガ　内田 洋一

コベリティ日本支社　安竹 由起夫

アジェンダ

- ソースコード静的解析ツールができること
  - 解析エンジンの動き
  - こんなバグ見つけてます

- セガ社の取り組み
  - 導入のきっかけ…
    - 自動化のメリット
  - 社内運用あれこれ
    - 各プラットフォームへの対応

アジェンダ

- ソースコード静的解析ツールができること
  - 解析エンジンの動き
  - こんなバグ見つけてます

- セガ社の取り組み
  - 導入のきっかけ…
    - 自動化のメリット
  - 社内運用あれこれ
    - 各プラットフォームへの対応

このコードにバグがありますか？

```
char *p;
if(x == 0){
  p = foo();
} else {
  p = 0;
}

if(x != 0){
  *p;
} else {
  ...;
}
return;
```

静的解析ツールができること

- ソースコードから直接不具合を検出
  - 動的テストに匹敵するシミュレーション
  - 実行する必要がないので、テスト環境不要

静的解析ツールができること

- テスト工程の無駄な手戻り作業を防ぐ！
  - 全自動、全パス（結合レベル）での解析
  - リソースリーク、メモリ破壊などを起こす箇所を検出

自分で自分を褒めたくなるフィードバック

Check in

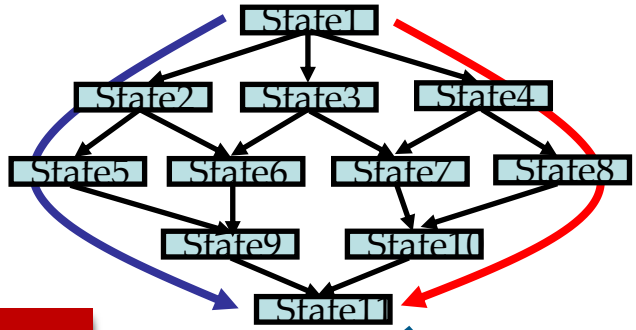
Check out

開発者


```
562   if (len == 0)
563       buf = NULL;
564   else
565   {
566       buf = malloc(len);
567
568       if (buf == NULL) {
569           read_error = TRUE;
570           return (READSTAT);
571       }
572   }
575   /* productA V1.0 */
576   rs_read_chars(inf, buf, len);
577
```

構成管理

静的解析



通知

解析

不具合レポート

不具合箇所の抽出

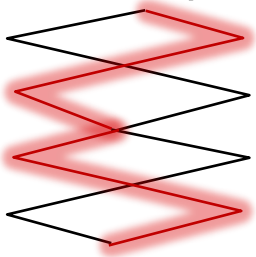
```
long int SomeFunction();
/* int OtherFunction(); */

/* int */ CallingFunction()
{
    long int test1;
    register /* int */ test2;

    test1 = SomeFunction();
    if (test1 > 0)
        test2 = 0;
    else
        test2 = OtherFunction();

    return test2;
}
```

不具合箇所に至るパスの抽出



coverity

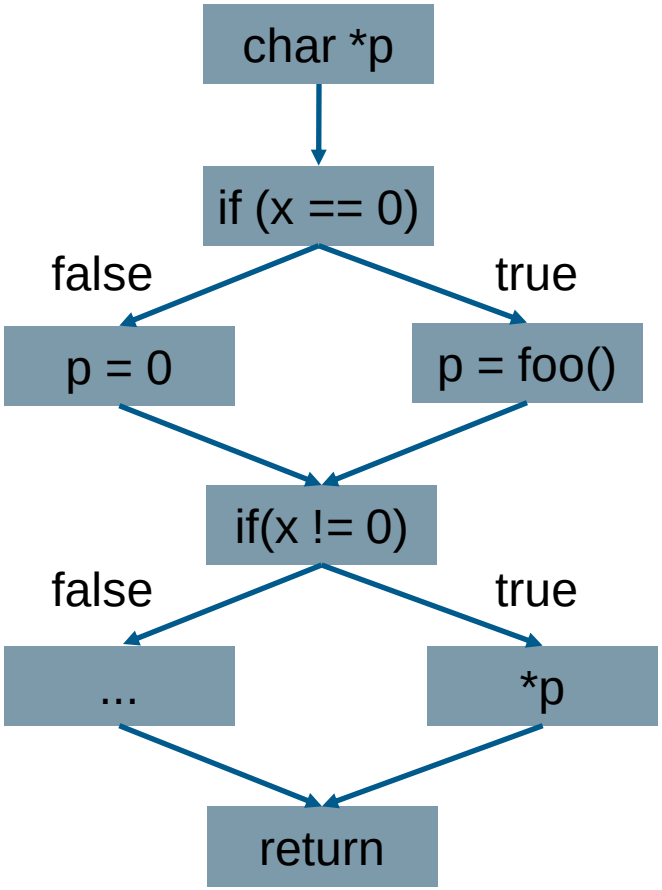
静的解析エンジンの動き

このコードにバグがありますか？

```
char *p;
if(x == 0){
  p = foo();
} else {
  p = 0;
}

if(x != 0){
  *p;
} else {
  ...;
}
return;
```

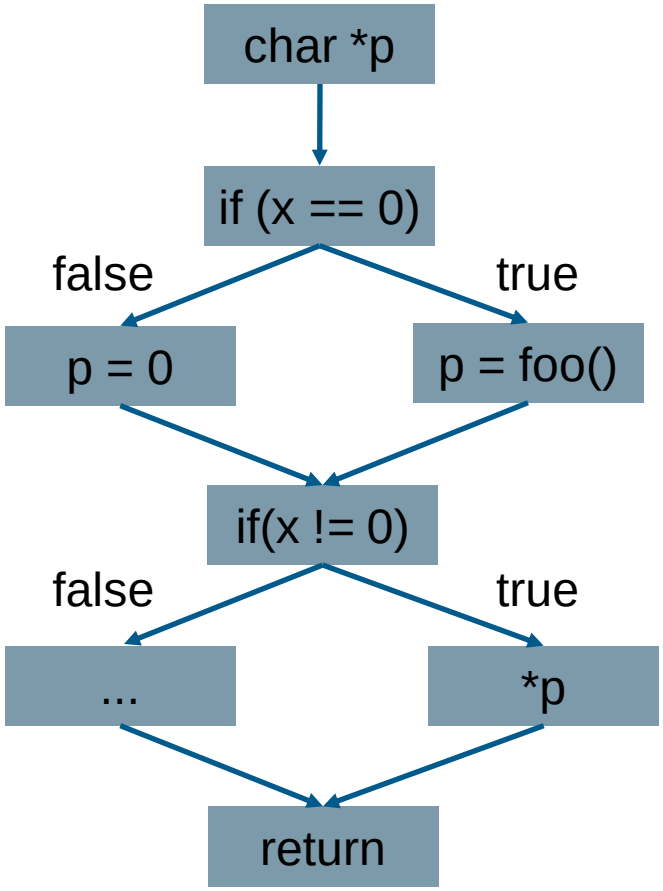
パス探索を行うと…

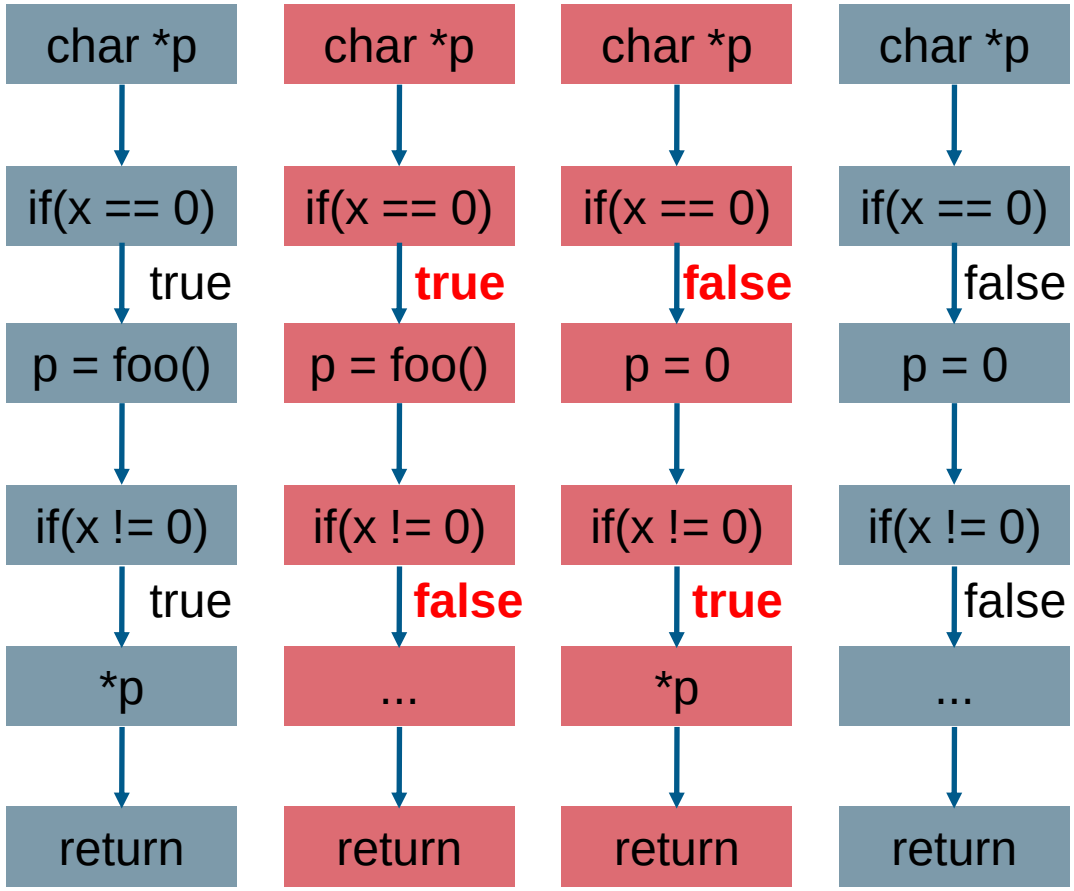



```
char *p;
if(x == 0){
  p = foo();
} else {
  p = 0;
}

if(x != 0){
  *p;
} else {
  ...;
}
return;
```

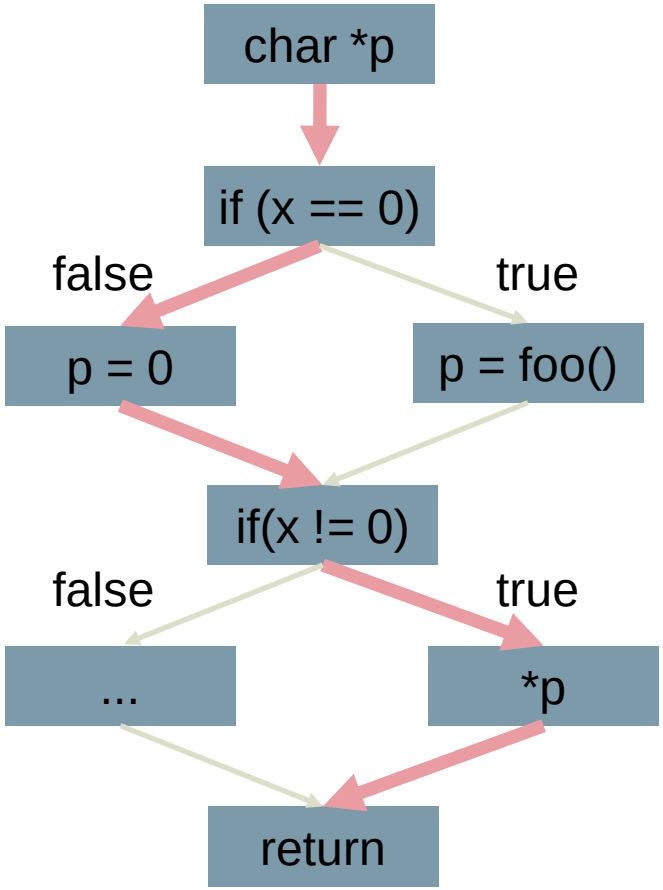
4つのパスが…、でも…

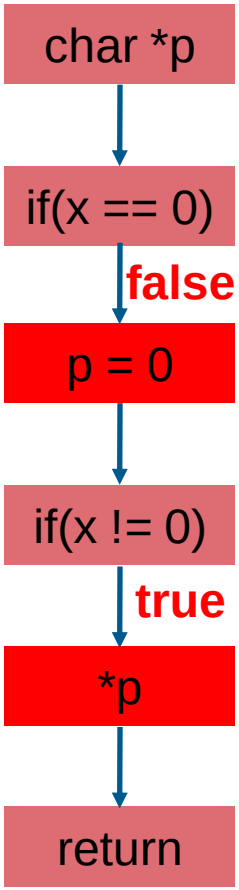




coverity

このコードから検出できるバグは1つ※1





※1 foo() のソースコードが存在しないと仮定

coverity

関数コールグラフを含む解析例

不具合コード

```
562   if (len == 0)
563       buf = NULL;
564   else
565   {
566       buf = malloc(len);
567
568       if (buf == NULL) {
569           read_error = TRUE;
570           return (READSTAT);
571       }
572   }
575   /* productA V1.0 */
576   rs_read_chars(inf, buf, len);
577
```

修正済みコード

```
562   /*
563   if
564     /
565     b
566
567     r
568   }

579   /*
580   /*
581   /*
582   rs_read_chars(inf, buf, len);
```

不具合コード

NULLの間接参照につながる、
パスと値がコード中に存在！
（直接 NULL を代入するコードが存在）

共通ライブラリコード

```
97  rs_read(FILE *inf, void *ptr, size_t size)
98  {
                    ….略….
102     if (encread(ptr, size, inf) != size)
```

```
317  encread(char *start, size_t size, FILE *inf)
318  {
319     char *e1, *e2, fb;
                    …略…
                ここでNULLの間接参照！
326     if ((read_size = fread(start,1,size,inf)) == 0) {
```

関数コールグラフを含む解析例

不具合コード

```
562   if (len == 0)
563       buf = NULL;
564   else
565   {
566       buf = malloc(len);
567
568       if (buf == NULL) {
569           read_error = TRUE;
570           return (READSTAT);
571       }
572   }
575   /* productA V1.0 */
576   rs_read_chars(inf, buf, len);
577
```

修正済みコード

```
562   /* product B */
563   if (len == 0) {
564     /* DO NOT SET NULL!!! */
565     buf = "default.txt";
566
567     return (READSTAT);
568   }
                … 略…
579   /* product B V1.0 */
580   /* reuse rs_read_chars */
581   /* original in productA V1.0 */
582   rs_read_chars(inf, buf, len);
```

修正済み ○

NULLの間接参照につながるパス、値がコードに存在しない問題なし！

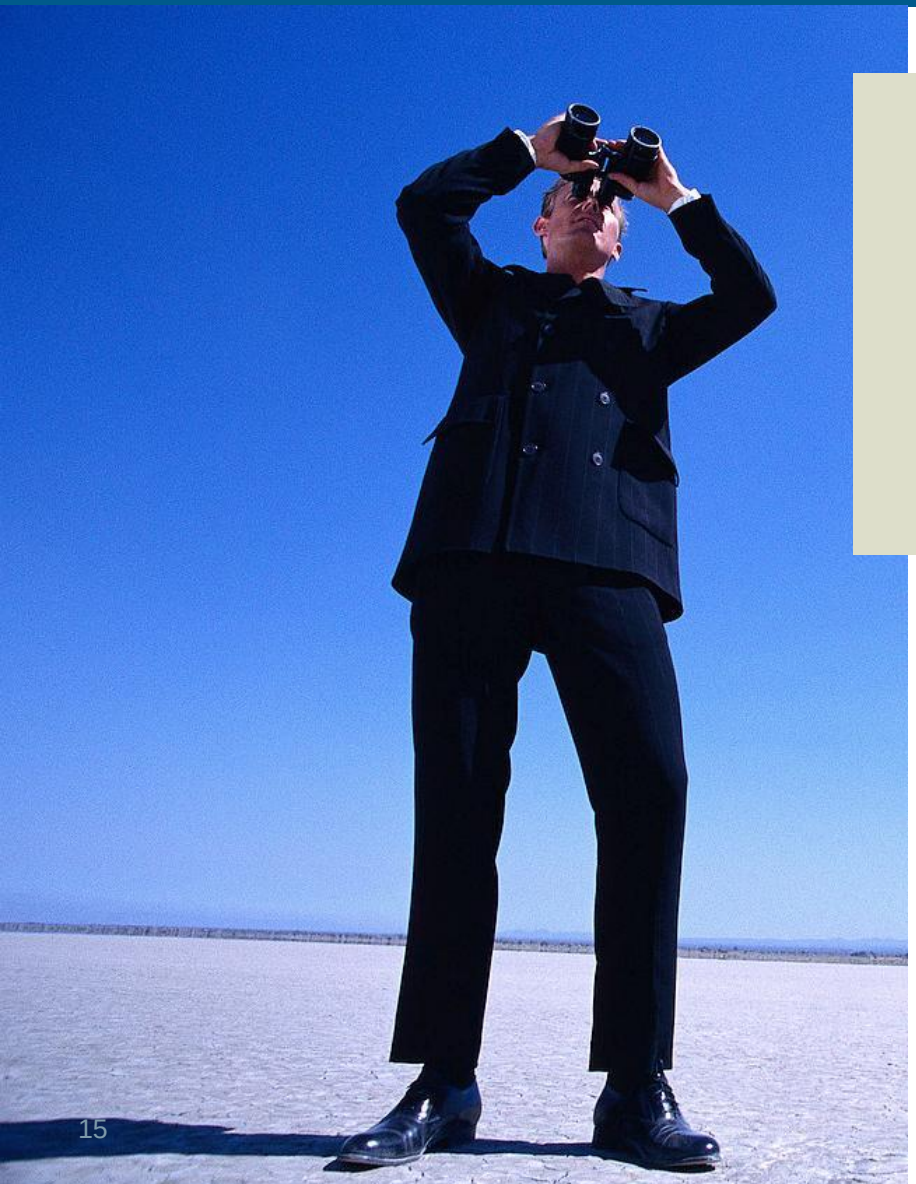
共通ライブラリコード

```
97  rs_read(FILE *inf, void *ptr, size_t size)
98  {
                ….略….
102     if (encread(ptr, size, inf) != size)
```

```
317  encread(char *start, size_t size, FILE *inf)
318  {
319     char *e1, *e2, fb;
                …略…

326     if ((read_size = fread(start,1,size,inf)) == 0) {
```

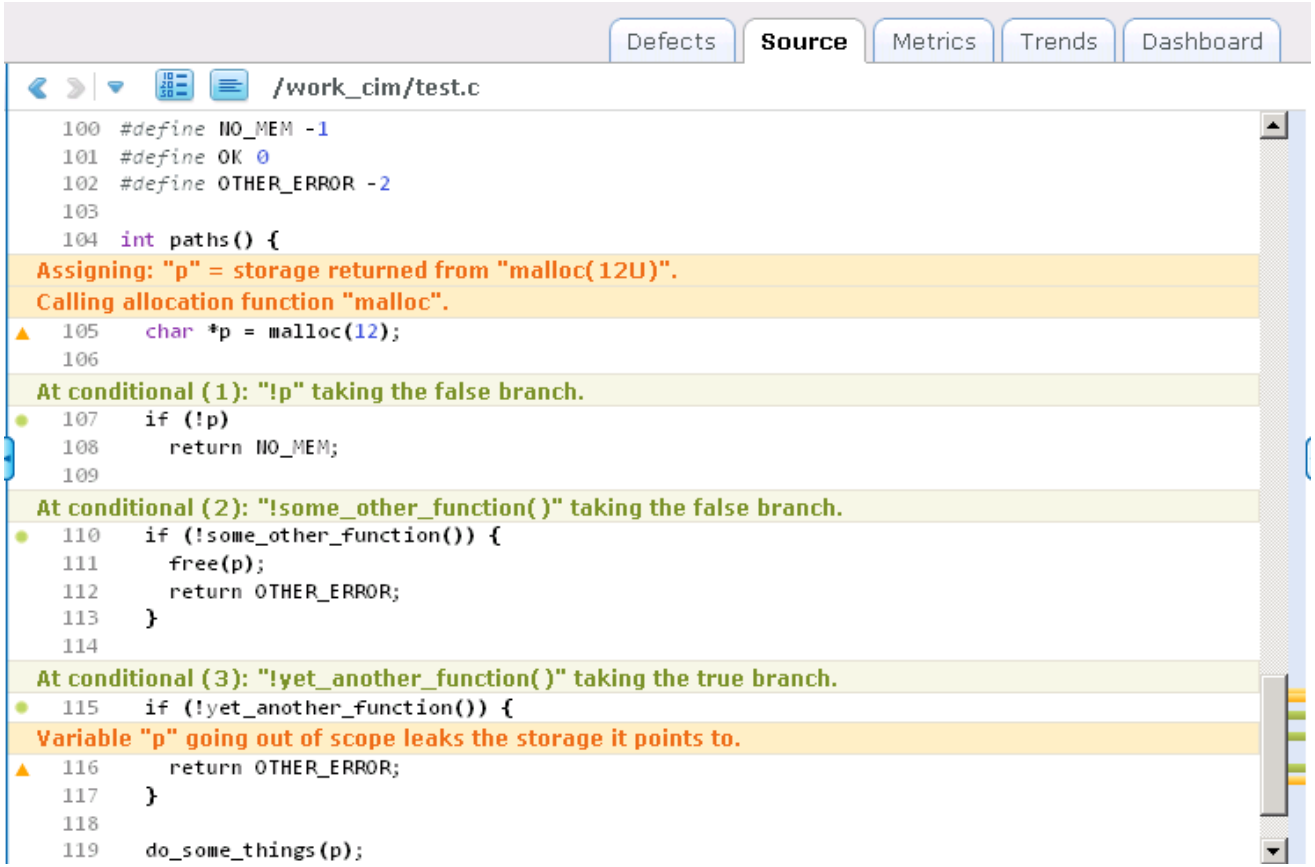
こんなバグ見つけてます。



- トライアル実績(～2009):
  - 約90Mステップ
- 日本顧客:
  - 約70社

coverity

不具合レポート



- 緑色のメッセージは、この不具合が発生するパス（分岐方向）を提示
- オレンジのメッセージは、不具合の発生箇所を提示

coverity

それ違う関数！

```
15    memcpy(handleKey, NULL, sizeof(struct HKey));
```

```
12          struct HKey handleKey;
13
14
A null pointer is passed to function "memcpy", which dereferences it.
A null pointer is passed to function "memcpy", which dereferences it. (Deref assumed on the basis of 'nonnull' parameter attribute.)
15          memcpy(&handleKey, NULL, sizeof(struct HKey));
16
```

NULL の間接参照
(FORWARD_NULL)

coverity

どんな不具合が見つかっているか？

	チェッカー名	チェック内容	CWE	リスク
1	**OVERRUN_STATIC**	配列の境界外のメモリ操作	119 125	高
2	**UNINIT_CTOR**	初期化されていないクラスまたは構造体の非静的データメンバーが存在	457	高
3	**FORWARD_NULL**	NULLを保持またはNULLチェックした変数がその後のパスで間接参照される箇所	476	中
4	**UNINIT**	初期化されていない変数	457	高
5	**STRING_OVERFLOW**	セキュリティ脆弱性に関する潜在的な文字列オーバーフロー	120	低
6	**UNUSED_VALUE**	関数呼び出しから割り当てられた値が利用されていない	563	低
7	**RESOURCE_LEAK**	メモリ・ファイルなどのリソースがスコープを外れリーク	403 404	高
8	**UNREACHABLE**	到達する制御フローが存在しないため実行されないコード	561	中
9	**CHECKED_RETURN**	統計的にみて、関数戻り値のチェックが一部で欠落	252	中
10	**DEADCODE**	分岐条件のために到達できず、実行されないコード	561	中

ありがちですが。

```
#define EK_TABLE_LOW 1
#define EK_TABLE_HIGH 15
…
if(ldx <= EK_TABLE_LOW && ldx > EK_TABLE_HIGH){
}
}
…
```

```
10
Assigning: "ldx" = "1".
11  ldx=1;
12
13
On this path, the condition "ldx > 15" cannot be true.
After this line, the value of "ldx" is equal to 1.
Noticing condition "ldx <= 1".
14  if(ldx <= EK_TABLE_LOW && ldx > EK_TABLE_HIGH){
Execution cannot reach this statement "ldy = 1;".
15      ldy = EK_TABLE_LOW;
16  } else {
17      ldy=0;
18  }
```

論理的に到達できないコード
（DEADCODE）

単純なミス、コードの流用時に多し。

```
1347  …
1348  YYCODETYPE yymajor;
1349  yyStackEntry *yytos = &pParser->yystack[pParser->yyidx];
1350
1351  if( pParser->yyidx<0 ) return 0;
1352 #ifndef NDEBUG
1353  if( yyTraceFILE && pParser->yyidx>=0 ){
1354    fprintf(yyTraceFILE,"%sPopping %s¥n",
1355      yyTracePrompt,
1356      yyTokenName[yytos->major]);
1357  }
```

チェックの場所が悪く、負の数を使用（REVERSE_NEGATIVE）

```
1347  static int yy_pop_parser_stack(yyParser *pParser){
1348    YYCODETYPE yymajor;
1349    yyStackEntry *yytos = &pParser->yystack[pParser->yyidx];
Using "pParser->yyidx" as index to array "pParser->yystack".
1350
1351    if( pParser->yyidx<0 ) return 0;
You might be using variable "pParser->yyidx" before verifying that it is >= 0.
1352  #ifndef NDEBUG
```

NULLチェックでも同様なパターンが

```
737 void EC_POINT_clear_free(EC_POINT *point)
738         {
739         if (!point) return;
740
741         if (point->meth->point_clear_finish != 0)
742                 point->meth->point_clear_finish(point);
743         else if (point->meth != NULL &&
                         point->meth->point_finish != 0)
744                 point->meth->point_finish(point);
```

```
739         if (!point) return;
740
```

Directly dereferencing pointer "point->meth".

```
741         if (point->meth->point_clear_finish != 0)
742                 point->meth->point_clear_finish(point);
```

Dereferencing "point->meth" before a null check.

```
743         else if (point->meth != NULL && point->meth->point_finish != 0)
744                 point->meth->point_finish(point);
745         OPENSSL_cleanse(point, sizeof *point);
746         OPENSSL_free(point);
747         }
748
```

パスが複雑になるにつれて…

```
625 …
626 char str[128];
627
628 if (buf =
629     retur
630 if (off)
631 {
632     if (c
633
634     memse
635     if (B
636
637 }
638 if (BIO_p
639     retur
640 for (i=0;
641 {
642     if ((
643     {
644
645
646 …
```

```
Assigning: "off" = "128".
633                     off=128;
634                 memset(str,' ',off);
分岐条件 (4): "BIO_write(fp, str, off) <= 0" は false に分岐しました。
635                 if (BIO_write(fp, str, off) <= 0)
636                     return 0;
637                 }
638
分岐条件 (5): "BIO_printf(fp, &"%s", name) <= 0" は false に分岐しました。
639             if (BIO_printf(fp,"%s", name) <= 0)
640                 return 0;
641
分岐条件 (6): "i < len" は true に分岐しました。
642             for (i=0; i<len; i++)
643                 {
分岐条件 (7): "i % 15U == 0U" は true に分岐しました。
644                 if ((i%15) == 0)
645                     {
646                     str[0]='\n';
Overrunning static array "&str[1]" of size 128 bytes by passing it to a function
argument "off + 4" at byte position 132.
647                     memset(&(str[1]),' ',off+4);
```

2つのリソース

```
3025    tpltname =    pathsearch(lemp->argv0,templatename,0);
3026  }
3027  if( tpltname==0 ){
3030    lemp->errorcnt++;
3031    return 0;
3032  }
3033  in = fopen(tpltname,"rb");
3034  if( in==0 ){
3036    lemp->errorcnt++;
3037    return 0;
3038  }
```

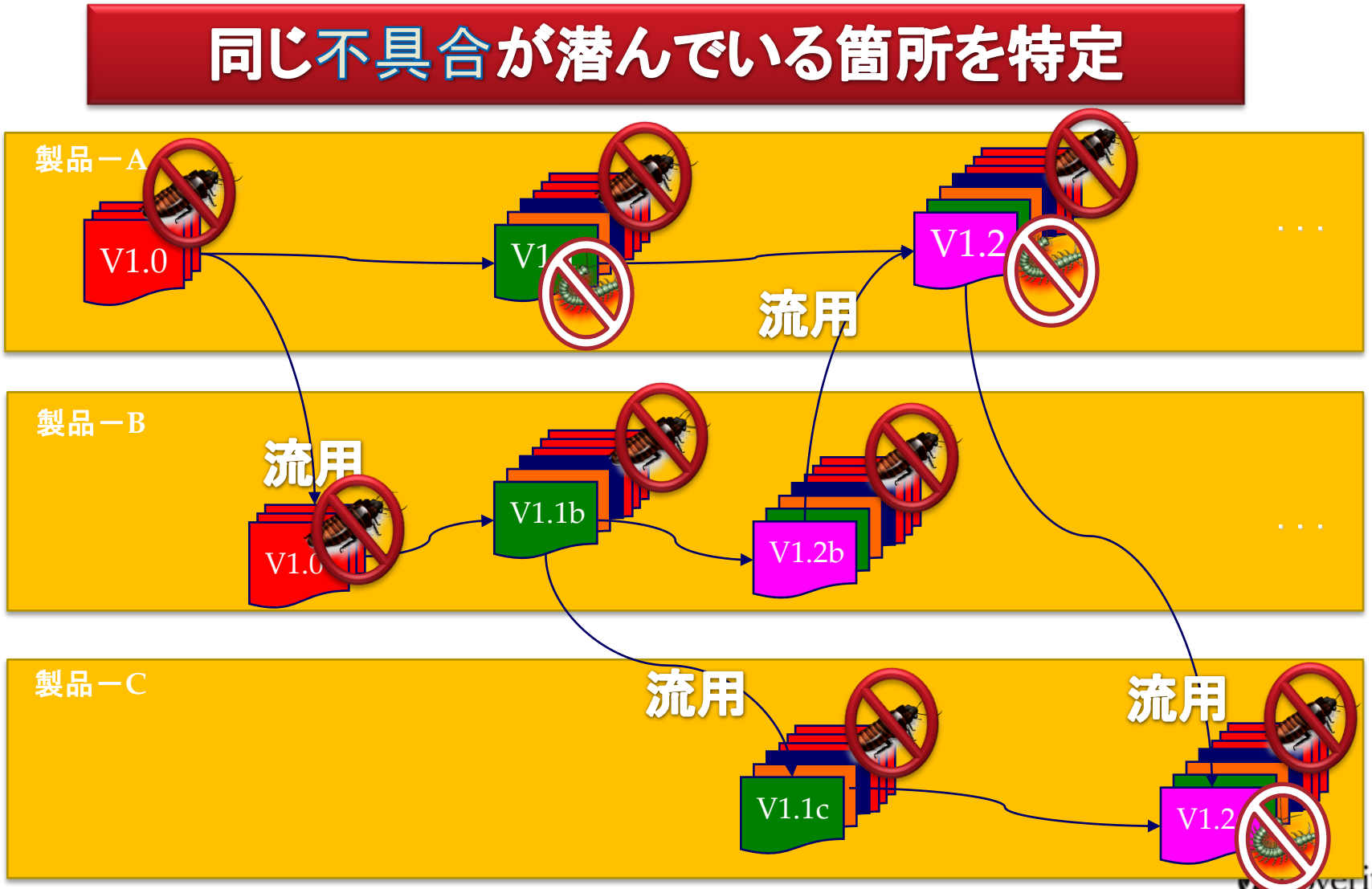
```
2905  PRIVATE char *pathsearch(argv0,name,modemask)
 …
2929    path = (char *)malloc( strlen(pathlist)+strlen(name)+2 );
 …
2944  return path;
2945}
```

実際のレポートを
見てみましょう

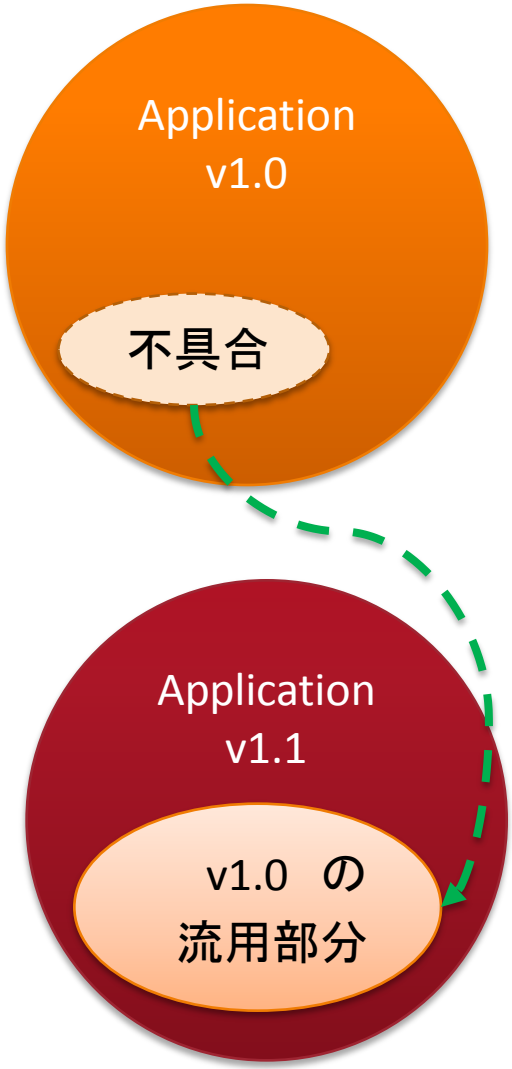
静的解析の今。派生開発への応用。

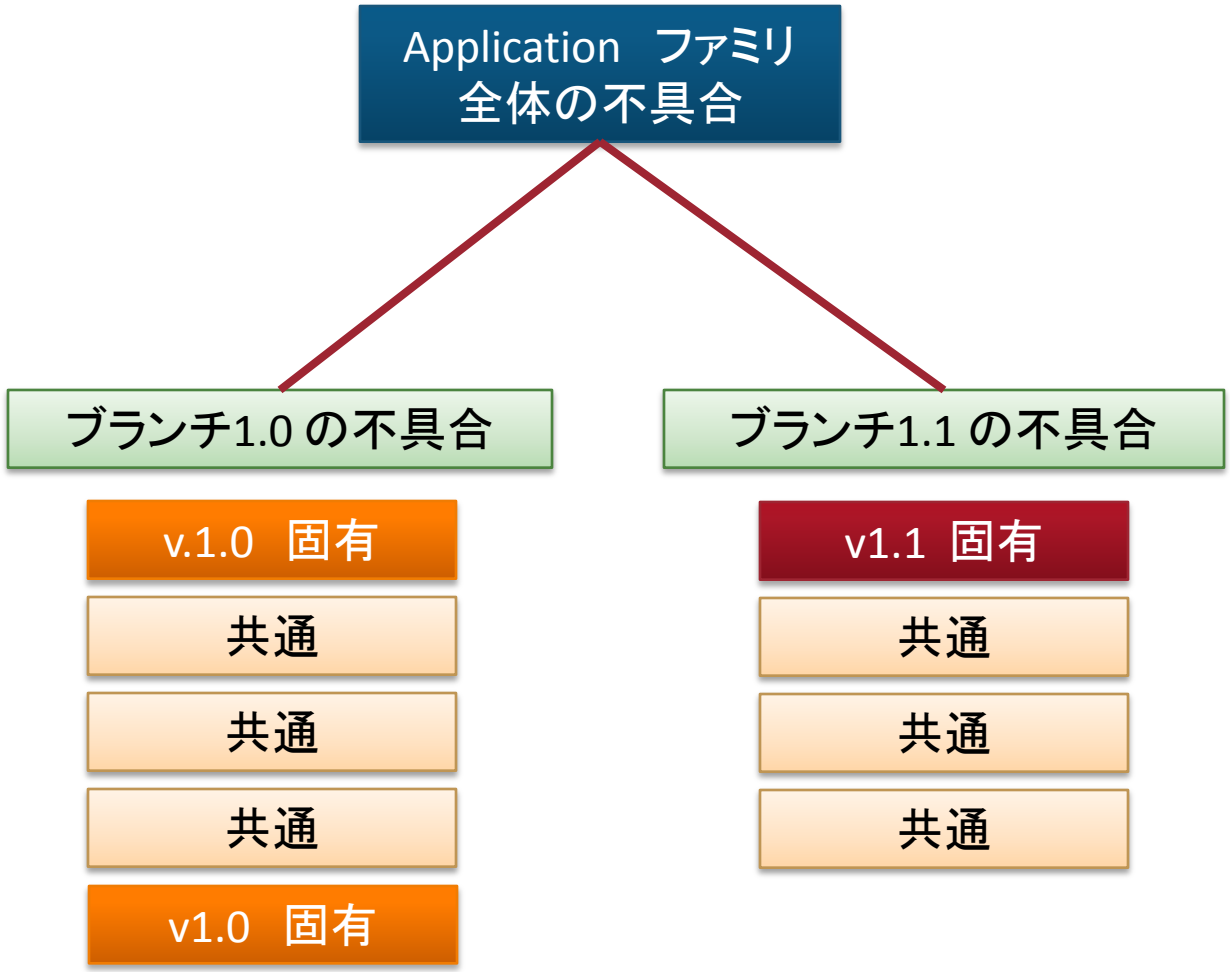
- 複数ブランチ、複数ターゲットへの対応
- 新しい優先度の設定方法

複数コードベースの解析結果を走査



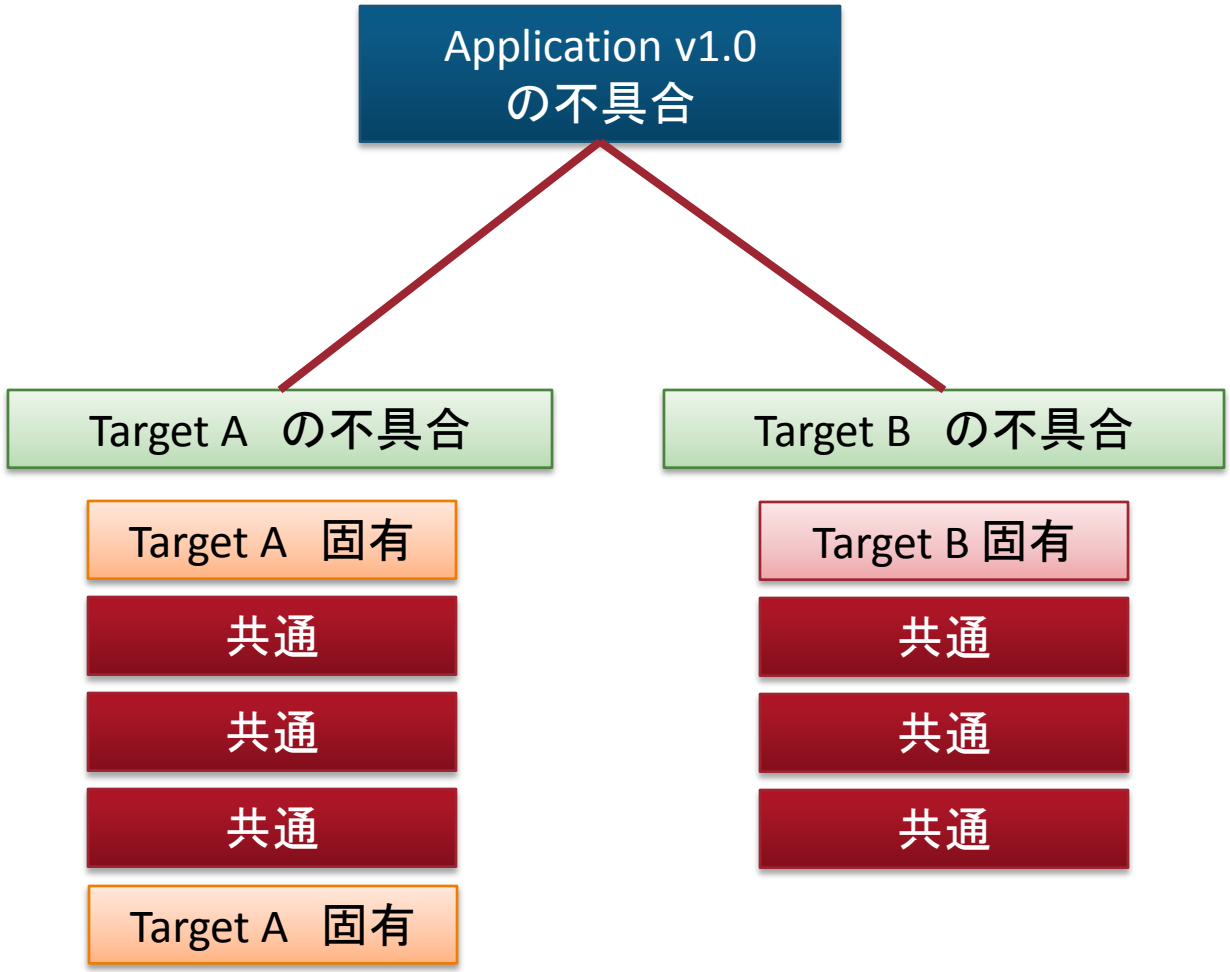
流用するソースコードが存在



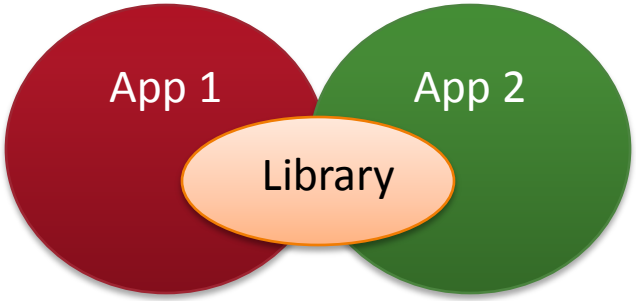


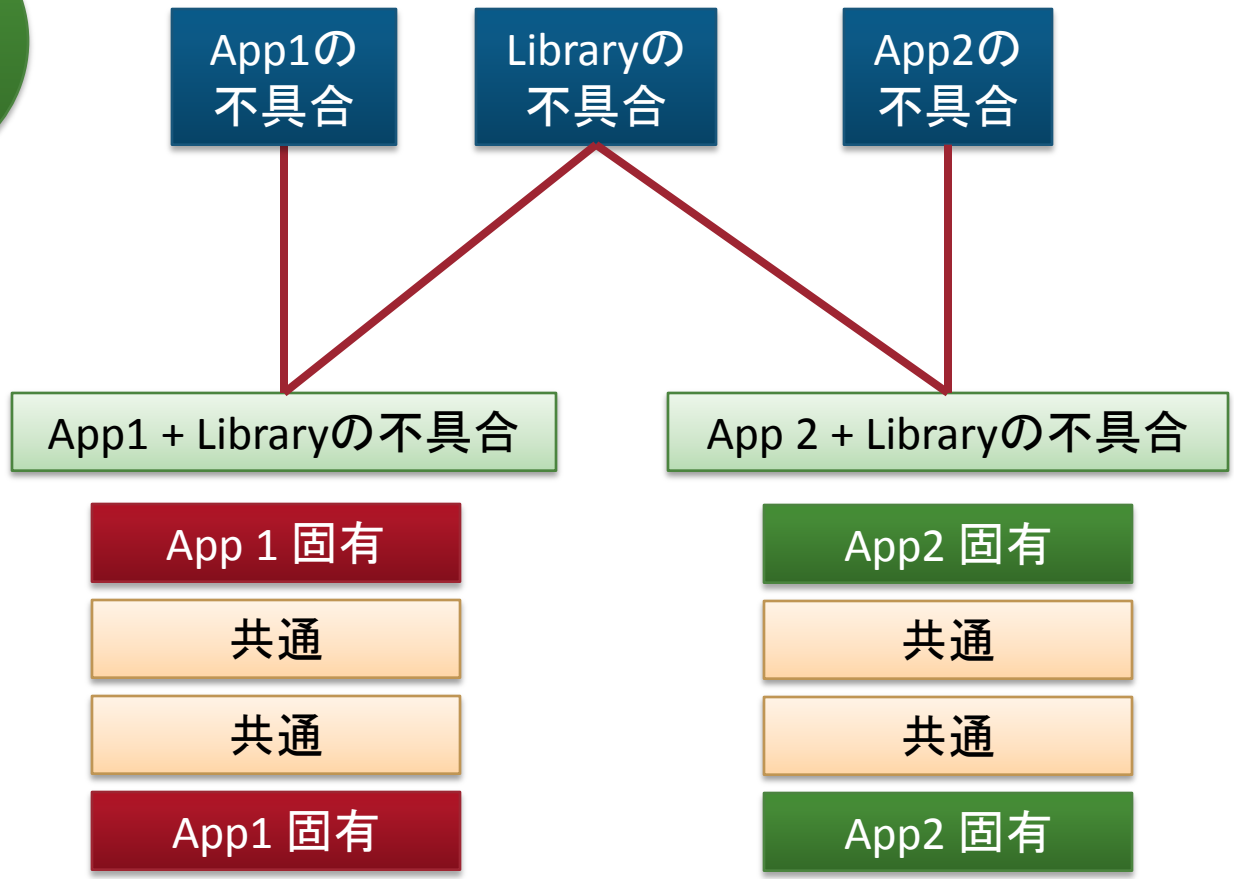
複数ターゲットの場合


Application v1.0
Target A
Target B


Application v1.0
の不具合
Target A の不具合
Target B の不具合
Target A 固有
共通
共通
共通
Target A 固有
Target B 固有
共通
共通
共通

共通ライブラリが存在する場合





libtiff

coverity™ a higher code™

バージョン (リリース年)	SLOC	関数の数	パス数	総検知不具合数	固有の不具合数
3.8.2 (2007)	70,716	834	217,266	70	6
3.9.1 (2009)	79,645	927	263,961	78	1
3.9.3 (2010)	80,452	932	283,495	76	0
4.0.0 beta 6 (2010)	87,244	1,052	304,821	68	13

全バージョン共通の不具合数： 44 件 (v3.8.2 からそのまま)

- リソースリーク、NULLチェックミス、解放後の利用、未初期化

平均不具合増加数：1,000 行あたり、0.6件

- このプロジェクトでは、1000行コードが増えると約0.6 件不具合が発生する

openSSL（ターゲット毎の比較）

coverity™ a higher code™

ターゲット	SLOC	関数の数	パス数	総検知不具合数	固有の不具合数
0.9.8i Linux(x86)	211,130	5,000	776,930	156	8
0.9.8i Windows	199,692	5,121	763,458	148	0

全ターゲット共通の不具合数： 148件

- パスがほとんど同じため、不具合の共通性は高い

Linux 側固有の不具合：8件

- デッドコード、NULLチェック箇所のミス、外部入力データの未チェック

静的解析ツールができること

- **静的解析は、結合レベルでのシュミレーションでこそ、自動化の威力を発揮する**
- **ソースコードの流用は不具合の伝播といえる**
  - 出荷後はテストされる機会が減り、不具合を含むコードが流用される可能性が高い
- **コードが追加されれば、不具合も混入する**
  - 成熟したオープンソースコード(C/C++)であっても、1,000行あたり約0.6件の不具合が新規で混入する
  - 商用コードベースの平均値は、1.5件/KSLOC。
- **ツールに検知された不具合は、対処の優先度決定が重要**
  - リスクが高い不具合(手動のテストで発見しにくい、システムを止めてしまう)
  - 出荷済みのコードと新規コードの不具合を区別

アジェンダ

- ソースコード静的解析ツールができること
  - 解析エンジンの動き
  - こんなバグ見つけてます

- セガ社の取り組み
  - 導入のきっかけ…
    - 自動化のメリット
  - 社内運用あれこれ
    - 各プラットフォームへの対応

Questions