Static Error Detection Using Semantic Inconsistency Inference

Isil Dillig, Thomas Dillig, Alex Aiken
Computer Science Department, Stanford University

June 13, 2007

Source-Sink Errors

- Many state-of-the-art static analysis tools target a class of errors we call source-sink errors.

Source-Sink Errors

- Many state-of-the-art static analysis tools target a class of errors we call source-sink errors.
- These errors arise when a distinguished "source" reaches a distinguished "sink".

Source-Sink Errors

- Many state-of-the-art static analysis tools target a class of errors we call source-sink errors.
- These errors arise when a distinguished "source" reaches a distinguished "sink".
- Typical examples of source sink errors include:
  - -Does a null pointer reach a dereference?
  - -Does a tainted input reach a security-critical operation?
  - -Does a closed file reach a read operation?

Source-Sink Errors

- Many state-of-the-art static analysis tools target a class of errors we call source-sink errors.
- These errors arise when a distinguished “source” reaches a distinguished “sink”.
- Typical examples of source sink errors include:
  - -Does a null pointer reach a dereference?
  - -Does a tainted input reach a security-critical operation?
  - -Does a closed file reach a read operation?
- Detection of these errors requires finding a “feasible” path between the source and the sink.

Inconsistency Errors

- A complementary approach for finding errors is inconsistency detection ("Bugs as Deviant Behavior", Engler et al.).

Inconsistency Errors

- A complementary approach for finding errors is inconsistency detection ("Bugs as Deviant Behavior", Engler et al.).
- A prototypical example:

```
if(x) a=*x;
...
b=*x;
```

Inconsistency Errors

- A complementary approach for finding errors is inconsistency detection (“Bugs as Deviant Behavior”, Engler et al.).
- A prototypical example:

```
if(x) a=*x;
...
b=*x;
```

- Inconsistency detection can be seen as a variation of type inference.

Inconsistency Errors

- A complementary approach for finding errors is inconsistency detection ("Bugs as Deviant Behavior", Engler et al.).
- A prototypical example:

```
if(x) a=*x;
...
b=*x;
```

- Inconsistency detection can be seen as a variation of type inference.
- The above program would not type check because x cannot have both `maybe-null` and `non-null` types.

Why is Inconsistency Detection Useful?

Why is Inconsistency Detection Useful?

- Source-sink analyzers assume closed programs. In open programs, sources can come from “the outside”.

Why is Inconsistency Detection Useful?

- Source-sink analyzers assume closed programs. In open programs, sources can come from “the outside”.
- Inconsistency errors are more local and hence easier to inspect and understand.

Why is Inconsistency Detection Useful?

- Source-sink analyzers assume closed programs. In open programs, sources can come from "the outside".
- Inconsistency errors are more local and hence easier to inspect and understand.
- Source-sink errors lie on a single program path. Inconsistencies can involve multiple program paths.

Defining Inconsistency

Defining Inconsistency

- An inconsistency error arises if two equivalent expressions are used in ways that indicate contradictory beliefs of the programmer.

Defining Inconsistency

- An inconsistency error arises if two equivalent expressions are used in ways that indicate contradictory beliefs of the programmer.
- Question #1: When are expressions equivalent?

Defining Inconsistency

- An inconsistency error arises if two equivalent expressions are used in ways that indicate contradictory beliefs of the programmer.
- Question #1: When are expressions equivalent?
- Question #2: How do we know if two uses indicate contradictory beliefs of the programmer?

Formalizing Inconsistencies

$$\textit{Function } F ::= \quad \textit{define } f(x_1, \ldots, x_n) = s$$

Formalizing Inconsistencies

$$
\begin{aligned}
\textit{Function } F ::= &\quad \textit{define } f(x_1, ..., x_n) = s \\
\textit{Statement } S ::= &\quad x \leftarrow^{\rho} C_i
\end{aligned}
$$

Formalizing Inconsistencies

$$
\begin{aligned}
\textit{Function } F ::= &\quad \textit{define } f(x_1, \ldots, x_n) = s \\
\textit{Statement } S ::= &\quad x \leftarrow^{\rho} C_i \\
&\quad | \texttt{check}^{\rho}\ x = C_i
\end{aligned}
$$

Formalizing Inconsistencies

$$
\begin{aligned}
\textit{Function } F &::= \quad \textit{define } f(x_1, \ldots, x_n) = s \\
\textit{Statement } S &::= \quad x \leftarrow^{\rho} C_i \\
&\quad\ |\, \texttt{check}^{\rho}\ x = C_i \\
&\quad\ |\, \textit{if}^{\rho}\ (x = C_i)\ s_1\ \textit{else}\ s_2
\end{aligned}
$$

Formalizing Inconsistencies

$$
\begin{array}{ll}
\textit{Function } F ::= & \textit{define } f(x_1, ..., x_n) = s \\
\textit{Statement } S ::= & x \leftarrow^{\rho} C_i \\
& |\texttt{check}^{\rho}\ x = C_i \\
& |\textit{if}^{\rho}\ (x = C_i)\ s_1\ \textit{else}\ s_2 \\
& |\ x \leftarrow^{\rho} y |\ f(x_1, ..., x_n)^{\rho}\ |\ s_1;^{\rho} s_2
\end{array}
$$

Formalizing Inconsistencies

$$
\begin{aligned}
\textit{Function } F ::= \quad & \textit{define } f(x_1, ..., x_n) = s \\
\textit{Statement } S ::= \quad & x \leftarrow^{\rho} C_i \\
& |\texttt{check}^{\rho}\ x = C_i \\
& |\textit{if}^{\rho}\ (x = C_i)\ s_1\ \textit{else}\ s_2 \\
& |\ x \leftarrow^{\rho} y |\ f(x_1, ..., x_n)^{\rho}\ |\ s_1 ;^{\rho} s_2
\end{aligned}
$$

- Constructors model type-state properties.
  e.g. `locked/unlocked`, `null/non-null`, `tainted/clean` etc.

Formalizing Inconsistencies

$$
\begin{array}{rl}
\textit{Function } F ::= & \textit{define } f(x_1, ..., x_n) = s \\
\textit{Statement } S ::= & x \leftarrow^{\rho} C_i \\
 & |\texttt{check}^{\rho}\ x = C_i \\
 & |\textit{if}^{\rho}\ (x = C_i)\ s_1\ \textit{else}\ s_2 \\
 & |\ x \leftarrow^{\rho} y |\ f(x_1, ..., x_n)^{\rho}\ |\ s_1 ;^{\rho} s_2
\end{array}
$$

- Constructors model type-state properties.
  e.g. `locked/unlocked`, `null/non-null`, `tainted/clean` etc.
- The only sources are constructor assignments; and the only sinks are `check` statements.

Formalizing Inconsistencies

$$
\begin{array}{ll}
\textit{Function } F ::= & \textit{define } f(x_1, \ldots, x_n) = s \\
\textit{Statement } S ::= & x \leftarrow^{\rho} C_i \\
& |\texttt{check}^{\rho}\ x = C_i \\
& |\textit{if}^{\rho}\ (x = C_i)\ s_1\ \textit{else}\ s_2 \\
& |\ x \leftarrow^{\rho} y |\ f(x_1, \ldots, x_n)^{\rho}\ |\ s_1 ;^{\rho} s_2
\end{array}
$$

- Constructors model type-state properties.
  e.g. `locked/unlocked`, `null/non-null`, `tainted/clean` etc.
- The only sources are constructor assignments; and the only sinks are `check` statements.
- Program points ρ are unique identifiers for every statement in the program.

Null Pointer Example

- We want to express that null pointer dereferences are illegal.

Null Pointer Example

- We want to express that null pointer dereferences are illegal.
- Add constructors `null` and `non-null`.

Null Pointer Example

- We want to express that null pointer dereferences are illegal.
- Add constructors `null` and `non-null`.
- Instrument every pointer dereference with a `check` statement:

  For every dereference

  ```
  *x
  ```

  add a check:

  ```
  check(x=non-null)
  ```

Guards

- To define semantics of inconsistency, we need path constraints.

Guards

- To define semantics of inconsistency, we need path constraints.
- We represent constraints as boolean formulas and call them *guards*.

Guards

- To define semantics of inconsistency, we need path constraints.
- We represent constraints as boolean formulas and call them *guards*.
- Useful to differentiate between two kinds of guards:

Guards

- To define semantics of inconsistency, we need path constraints.
- We represent constraints as boolean formulas and call them *guards*.
- Useful to differentiate between two kinds of guards:

1. *Statement guards* γ^{ρ} that describe the conditions under which a statement ρ is reached.

Guards

- To define semantics of inconsistency, we need path constraints.
- We represent constraints as boolean formulas and call them *guards*.
- Useful to differentiate between two kinds of guards:

1. *Statement guards* γ^ρ that describe the conditions under which a statement ρ is reached.
2. *Constructor guards* $\Gamma^\rho(x)(j)$ which describe the conditions under which a variable x evaluates to constructor C_j.

Definition of Inconsistency

Consider two `check` statements $check^{\rho_0}(x = C_i)$ and $check^{\rho_1}(y = C_i)$.

Definition of Inconsistency

Consider two `check` statements $check^{\rho_0}(x = C_i)$ and $check^{\rho_1}(y = C_i)$.

An inconsistency error arises if:

Definition of Inconsistency

Consider two `check` statements $check^{\rho_0}(x = C_i)$ and $check^{\rho_1}(y = C_i)$.

An inconsistency error arises if:

1. $$x \cong y$$

Definition of Inconsistency

Consider two `check` statements $check^{\rho_0}(x = C_i)$ and $check^{\rho_1}(y = C_i)$.

An inconsistency error arises if:

1. $$x \cong y$$
   x *and* y *are semantically equivalent*.

Definition of Inconsistency

Consider two `check` statements $check^{\rho_0}(x = C_i)$ and $check^{\rho_1}(y = C_i)$.

An inconsistency error arises if:

1. $$x \cong y$$
   x *and* y *are semantically equivalent.*

2. $$\neg SAT(\gamma^{\rho_0} \wedge \bigvee_{j \neq i} \Gamma(x)(j))$$

Definition of Inconsistency

Consider two `check` statements $check^{\rho_0}(x = C_i)$ and $check^{\rho_1}(y = C_i)$.

An inconsistency error arises if:

1. $$x \cong y$$
   x *and* y *are semantically equivalent.*

2. $$\neg SAT(\gamma^{\rho_0} \wedge \bigvee_{j \neq i} \Gamma(x)(j))$$
   x *is guaranteed to be* C_i at ρ_0

Definition of Inconsistency

Consider two `check` statements $check^{\rho_0}(x = C_i)$ and $check^{\rho_1}(y = C_i)$.

An inconsistency error arises if:

1. $$x \cong y$$
   x *and* y *are semantically equivalent.*

2. $$\neg SAT(\gamma^{\rho_0} \wedge \bigvee_{j \neq i} \Gamma(x)(j))$$
   x *is guaranteed to be* C_i at ρ_0

3. $$SAT(\gamma^{\rho_1} \wedge \bigvee_{j \neq i} \Gamma(y)(j)))$$

Definition of Inconsistency

Consider two `check` statements $check^{\rho_0}(x = C_i)$ and $check^{\rho_1}(y = C_i)$.

An inconsistency error arises if:

1. $$x \cong y$$
   x *and* y *are semantically equivalent*.

2. $$\neg SAT(\gamma^{\rho_0} \wedge \bigvee_{j \neq i} \Gamma(x)(j))$$
   x *is guaranteed to be* C_i at ρ_0

3. $$SAT(\gamma^{\rho_1} \wedge \bigvee_{j \neq i} \Gamma(y)(j)))$$
   At ρ_1 it is possible for y to be C_j

Congruence

- The definition of congruence (or semantic equivalence) depends on the language and types of expressions.

Congruence

- The definition of congruence (or semantic equivalence) depends on the language and types of expressions.
- An especially problematic case is pointers.

Congruence

- The definition of congruence (or semantic equivalence) depends on the language and types of expressions.
- An especially problematic case is pointers.
- Given a points-to graph (V, E) and two pointer variables $v_1, v_2 \in V$, we define congruence as:

$$v_1 \cong v_2 \Leftrightarrow \forall v_3 \in V.(((v_1, v_3) \in E \Leftrightarrow (v_2, v_3) \in E)$$

Congruence

- The definition of congruence (or semantic equivalence) depends on the language and types of expressions.
- An especially problematic case is pointers.
- Given a guarded points-to graph (V, E) and two pointer variables $v_1, v_2 \in V$, we have:

$$v_1 \cong v_2 \Leftrightarrow \forall v_3 \in V.(((v_1, v_3)^{g_1} \in E \Leftrightarrow (v_2, v_3)^{g_2} \in E) \wedge g_1 \equiv g_2$$

Inconsistency Example

example.c

```
b = (p!=NULL)
q = p;
if(b)
  *p = 8;
*q = 4;
```

The Analysis

Inconsistency Example

example.c

```
b = (p!=NULL)
q = p;
if(b)
  *p = 8;
*q = 4;
```

The Analysis

if(p=null)
  b ← C_0
else
  b ← C_1;

Inconsistency Example

example.c

```
b = (p!=NULL)
q = p;
if(b)
  *p = 8;
*q = 4;
```

The Analysis

if(p=null)
  b ← C_0
else
  b ← C_1;

γ=(p=null)

Inconsistency Example

example.c

```
b = (p!=NULL)
q = p;
if(b)
  *p = 8;
*q = 4;
```

The Analysis

if(p=null)
  b ← C_0
else
  b ← C_1;

$\Gamma(b)(C_0)$=(p=null)

Inconsistency Example

example.c

```
b = (p!=NULL)
q = p;
if(b)
  *p = 8;
*q = 4;
```

The Analysis

if(p=null)
  b ← C_0
else
  b ← C_1;

γ=(p=non-null)

Inconsistency Example

example.c

```
b = (p!=NULL)
q = p;
if(b)
  *p = 8;
*q = 4;
```

The Analysis

if(p=null)
  b ← C_0
else
  b ← C_1;

$\Gamma(b)(C_1)$ = (p=non-null)

Inconsistency Example

example.c

```
b = (p!=NULL)
q = p;
if(b)
  *p = 8;
*q = 4;
```

The Analysis

if(p=null)
  b ← C_0
else
  b ← C_1;
q ← p;

Inconsistency Example

example.c

```
b = (p!=NULL)
q = p;
if(b)
  *p = 8;
*q = 4;
```

The Analysis

if(p=null)
  b $\leftarrow C_0$
else
  b $\leftarrow C_1$;
q $\leftarrow$ p;

$q \cong p$

Inconsistency Example

example.c

```
b = (p!=NULL)
q = p;
if(b)
  *p = 8;
*q = 4;
```

The Analysis

if(p=null)
  b ← C_0
else
  b ← C_1;
q ← p;
if(b=C_1)

Inconsistency Example

example.c

```
b = (p!=NULL)
q = p;
if(b)
  *p = 8;
*q = 4;
```

The Analysis

if(p=null)
  b $\leftarrow C_0$
else
  b $\leftarrow C_1$;
q $\leftarrow$ p;
if(b=C_1)

$\gamma = \Gamma(b)(C_1) = (\texttt{p=non-null})$

Inconsistency Example

example.c

```
b = (p!=NULL)
q = p;
if(b)
  *p = 8;
*q = 4;
```

The Analysis

```
if(p=null)
  b ← C0
else
  b ← C1;
q ← p;
if(b=C1)
  check(p=non-null);
```

Inconsistency Example

example.c

```
b = (p!=NULL)
q = p;
if(b)
  *p = 8;
*q = 4;
```

The Analysis

if(p=null)
  b ← C_0
else
  b ← C_1;
q ← p;
if(b=C_1)
  check(p=non-null); $\neg SAT(\gamma \wedge \Gamma(p)(\texttt{null}))$

Inconsistency Example

example.c

```
b = (p!=NULL)
q = p;
if(b)
  *p = 8;
*q = 4;
```

The Analysis

if(p=null)
  b ← C_0
else
  b ← C_1;
q ← p;
if(b=C_1)
  check(p=non-null);
check(q=non-null);

Inconsistency Example

example.c

```
b = (p!=NULL)
q = p;
if(b)
  *p = 8;
*q = 4;
```

The Analysis

if(p=null)
  b ← C_0
else
  b ← C_1;
q ← p;
if(b=C_1)
  check(p=non-null);
check(q=non-null); $SAT(\gamma \wedge \Gamma(q)(\texttt{null}))$

Inconsistency Example

example.c

```
b = (p!=NULL)
q = p;
if(b)
  *p = 8;
*q = 4;
```

The Analysis

if(p=null)
  b ← C_0
else
  b ← C_1;
q ← p; $q \cong p$
if(b=C_1)
  check(p=non-null); $\neg SAT(\gamma \wedge \Gamma(p)(\texttt{null}))$
check(q=non-null); $SAT(\gamma \wedge \Gamma(q)(\texttt{null}))$

Interprocedural Inconsistencies

- The definition of inconsistencies needs to be slightly modified in the interprocedural case.

Interprocedural Inconsistencies

- The definition of inconsistencies needs to be slightly modified in the interprocedural case.
- Consider these slightly different function definitions:

```
void f(int* x)
{
  if(x)
    *x = 2;
  g(x);
}
void g(int* y)
{
  *y=3;
}
```

```
void f'(int* x)
{
  *x=2;
  g'(x);
}

void g'(int* y)
{
  if(y)
    *y=3;
}
```

Interprocedural Inconsistencies

- The definition of inconsistencies needs to be slightly modified in the interprocedural case.
- Consider these slightly different function definitions:

```
void f(int* x)
{
  if(x)
    *x = 2;
  g(x);
}
void g(int* y)
{
  *y=3;
}
```

```
void f'(int* x)
{
  *x=2;
  g'(x);
}

void g'(int* y)
{
  if(y)
    *y=3;
}
```

Revised Definition for Interprocedural Inconsistencies

- Consider two `check` statements $check^{\rho_0}(x = C_i)$ and $check^{\rho_1}(y = C_i)$. An inconsistency error arises if:

$$
\begin{array}{ll}
1. & x \cong y \\
2. & \neg SAT(\gamma^{\rho_0} \wedge \bigvee_{j \neq i} \Gamma(x)(j)) \wedge InCaller(\rho_0) \\
3. & SAT(\gamma^{\rho_1} \wedge \bigvee_{j \neq i} \Gamma(x)(j))) \wedge InCallee(\rho_1)
\end{array}
$$

Implementation and Results

- We implemented a null dereference analysis for C with both a source-sink analyzer and an inconsistency detector using the SATURN infrastructure.

Implementation and Results

- We implemented a null dereference analysis for C with both a source-sink analyzer and an inconsistency detector using the SATURN infrastructure.
- We analyzed OpenSSH, OpenSSL, Samba, Sendmail, Pine, MPlayer, and the entire Linux kernel to find source-sink and inconsistency errors.

Implementation and Results

- We implemented a null dereference analysis for C with both a source-sink analyzer and an inconsistency detector using the SATURN infrastructure.
- We analyzed OpenSSH, OpenSSL, Samba, Sendmail, Pine, MPlayer, and the entire Linux kernel to find source-sink and inconsistency errors.
- We found 518 inconsistency errors and 77 source-sink errors with a false positve rate of 19.5%.

Implementation and Results

- We implemented a null dereference analysis for C with both a source-sink analyzer and an inconsistency detector using the SATURN infrastructure.
- We analyzed OpenSSH, OpenSSL, Samba, Sendmail, Pine, MPlayer, and the entire Linux kernel to find source-sink and inconsistency errors.
- We found 518 inconsistency errors and 77 source-sink errors with a false positve rate of 19.5%.
- Developers of Samba and Linux claim the errors found by our tool had not been detected by other static analysis tools, from which they receive regular checking.

Sample Linux Error Report

```
/* Linux, net/sctp/output.c, line 270 */
236 pmtu = ((packet->transport->asoc) ?
237   (packet->transport->asoc->pathmtu) :
238   (packet->transport->pathmtu));
...
269 if (sctp_chunk_is_data(chunk)) {
270   retval = sctp_packet_append_data(packet, chunk);
...
286 }

538 sctp_xmit_t sctp_packet_append_data (struct sctp_packet *packet,
539                                      struct sctp_chunk *chunk)
540 {
...
543 struct sctp_transport *transport = packet->transport;
...
545 struct sctp_association *asoc = transport->asoc;
...
562 rwnd = asoc->peer.rwnd;
```

Sample Linux Error Report

```
/* Linux, net/sctp/output.c, line 270 */
236 pmtu = ((packet->transport->asoc) ?
237    (packet->transport->asoc->pathmtu) :
238    (packet->transport->pathmtu));
...
269 if (sctp_chunk_is_data(chunk)) {
270   retval = sctp_packet_append_data(packet, chunk);
...
286 }

538 sctp_xmit_t sctp_packet_append_data (struct sctp_packet *packet,
539                                      struct sctp_chunk *chunk)
540 {
...
543 struct sctp_transport *transport = packet->transport;
...
545 struct sctp_association *asoc = transport->asoc;
...
562 rwnd = asoc->peer.rwnd;
```

Sample Linux Error Report

```
/* Linux, net/sctp/output.c, line 270 */
236 pmtu = ((packet->transport->asoc) ?
237   (packet->transport->asoc->pathmtu) :
238   (packet->transport->pathmtu));
...
269 if (sctp_chunk_is_data(chunk)) {
270   retval = sctp_packet_append_data(packet, chunk);
...
286 }

538 sctp_xmit_t sctp_packet_append_data (struct sctp_packet *packet,
539                                      struct sctp_chunk *chunk)
540 {
...
543 struct sctp_transport *transport = packet->transport;
...
545 struct sctp_association *asoc = transport->asoc;
...
562 rwnd = asoc->peer.rwnd;
```

Sample Linux Error Report

```
/* Linux, net/sctp/output.c, line 270 */
236 pmtu = ((packet->transport->asoc) ?
237    (packet->transport->asoc->pathmtu) :
238    (packet->transport->pathmtu));
...
269 if (sctp_chunk_is_data(chunk)) {
270    retval = sctp_packet_append_data(packet, chunk);
...
286 }

538 sctp_xmit_t sctp_packet_append_data (struct sctp_packet *packet,
539                                      struct sctp_chunk *chunk)
540 {
...
543 struct sctp_transport *transport = packet->transport;
...
545 struct sctp_association *asoc = transport->asoc;
...
562 rwnd = asoc->peer.rwnd;
```

Sample Linux Error Report

```
/* Linux, net/sctp/output.c, line 270 */
236 pmtu = ((packet->transport->asoc) ?
237   (packet->transport->asoc->pathmtu) :
238   (packet->transport->pathmtu));
...
269 if (sctp_chunk_is_data(chunk)) {
270   retval = sctp_packet_append_data(packet, chunk);
...
286 }

538 sctp_xmit_t sctp_packet_append_data (struct sctp_packet *packet,
539                                      struct sctp_chunk *chunk)
540 {
...
543 struct sctp_transport *transport = packet->transport;
...
545 struct sctp_association *asoc = transport->asoc;
...
562 rwnd = asoc->peer.rwnd;
```

Sample Linux Error Report

```
/* Linux, net/sctp/output.c, line 270 */
236 pmtu = ((packet->transport->asoc) ?
237   (packet->transport->asoc->pathmtu) :
238   (packet->transport->pathmtu));
...
269 if (sctp_chunk_is_data(chunk)) {
270   retval = sctp_packet_append_data(packet, chunk);
...
286 }

538 sctp_xmit_t sctp_packet_append_data (struct sctp_packet *packet,
539                                      struct sctp_chunk *chunk)
540 {
...
543 struct sctp_transport *transport = packet->transport;
...
545 struct sctp_association *asoc = transport->asoc;
...
562 rwnd = asoc->peer.rwnd;
```

OpenSSL Example

```
/* OpenSSL, e_chil.c line 1040 */
static int hwcrhk_rsa_mod_exp(BIGNUM *r, const BIGNUM *I,
               RSA *rsa, BN_CTX *ctx)
967  {
985   if ((hptr = RSA_get_ex_data(rsa, hndidx_rsa))!= NULL)
987   {
990     if(!rsa->n){
994     goto err;
995     }

997   /* Prepare the params */
998   bn_expand2(r, rsa->n->top); /* Check for error !!  */
      ...
1027 }
1028  else {
      ...
1039   /* Prepare the params */
1040   bn_expand2(r, rsa->n->top); /* Check for error !!  */
       ...
1080 }
```

OpenSSL Example

```
/* OpenSSL, e_chil.c line 1040 */
static int hwcrhk_rsa_mod_exp(BIGNUM *r, const BIGNUM *I,
               RSA *rsa, BN_CTX *ctx)
967  {
985   if ((hptr = RSA_get_ex_data(rsa, hndidx_rsa))!= NULL)
987   {
990     if(!rsa->n){
994     goto err;
995     }

997   /* Prepare the params */
998   bn_expand2(r, rsa->n->top); /* Check for error !!  */
      ...
1027 }
1028  else {
      ...
1039   /* Prepare the params */
1040   bn_expand2(r, rsa->n->top); /* Check for error !!  */
       ...
1080 }
```

OpenSSL Example

```
/* OpenSSL, e_chil.c line 1040 */
static int hwcrhk_rsa_mod_exp(BIGNUM *r, const BIGNUM *I,
               RSA *rsa, BN_CTX *ctx)
967  {
985   if ((hptr = RSA_get_ex_data(rsa, hndidx_rsa))!= NULL)
987   {
990     if(!rsa->n){
994     goto err;
995     }

997   /* Prepare the params */
998   bn_expand2(r, rsa->n->top); /* Check for error !!  */
      ...
1027 }
1028  else {
      ...
1039   /* Prepare the params */
1040   bn_expand2(r, rsa->n->top); /* Check for error !!  */
       ...
1080 }
```

Related Work

D. Engler, D. Chen, S. Hallem, A. Chou, and B. Chelf.
Bugs as deviant behavior: A general approach to inferring errors in systems code.
Operating Systems Review, 35(5):57–72, 2001.

B. Hackett and A. Aiken.
How is aliasing used in systems software?
In *Proceedings of the ACM International Symposium on Foundations of Software Engineering*, pages 69–80, 2006.

G. Necula, S. McPeak, and W. Weimer.
CCured: Type-safe retrofitting of legacy code.
In *Proc. of the Symp. on Principles of Prog. Languages*, pages 128–139, 2002.

D. Beyer, T. Henzinger, R. Jhala, and R. Majumdar.
Checking memory safety with Blast.
In *Proc. of the Conf. on Fundamental Approaches to Software Engineering*, pages 2–18, 2005.

R. Cartwright and M. Fagan.
Soft typing.
In *Proc. of the Conf. on Prog. Language Design and Implementation*, pages 278–292, 1991.

M. Faehndrich and K. Rustan M. Leino.
Declaring and checking non-null types in an object-oriented language.
In *Proc. of the Conf. on Object-Oriented Programing, Systems, Languages and Applications*, pages 302–312, 2003.

Thanks for listening!