Building Cyber Resilient Interfaces in z/OS

Elijah Swift, CISSP
IBM z/OS Secure Engineering

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Agenda

- What does Cyber Resiliency mean on z/OS?
- Vulnerability patterns on z/OS
- Program check analysis
- What not to say when writing Cyber Resilient software on z/OS

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

WHAT DOES CYBER RESILIENCY MEAN ON Z/OS?

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

What is Cyber Resiliency?

- Anticipate, withstand, recover from or adapt to adverse cyber incidents
- Preserve “System Integrity”
  - Prevents users from circumventing security mechanisms
- In z/OS, this means there is no way for an unauthorized problem program to:
  - Bypass store or fetch protection
  - Bypass RACF protection
  - Obtain control in an authorized state
- IBM will resolve any reported system integrity problem in supported releases

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

How many lines of code does it take to compromise all security and integrity on the z/OS solution stack?

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

One

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

What is an “authorized” program?

- Supervisor State (vs. Problem State)
- PSW Key 0-7 (vs. User Key 8-15)
  - Also known as “System Key”
- APF Authorization
  - Loaded from an APF–authorized library
  - Link–edited with authorization code AC=1

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

How does an attacker gain ”authorization”?

Many interfaces that gain authorization allow regular users to call or invoke them.

- SVC and PC routines
- APF authorized programs
  - Job step programs linked AC(1)
- Program Properties Table programs
- UNIX set-user-id and set-group-id programs

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Focus on the “Boundary”

Be careful when designing interfaces that meet the following criteria:

- Allow unauthorized requester
- Uses authorized service

- Unauthorized requester’s parameters are NOT to be trusted
  - Reference these in the caller’s key

Safe copy instructions shown below

MVCK – Move With Key
MVCSK – Move With Source Key
MVCDK – Move With Destination Key
MVCOS – Move With Optional Specifications

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Referencing User-Key Storage

Why not read or write to caller-specified storage while running in system key?

- User may pass in system key storage

- Key of the storage could have changed
  - Time of check to time of use problem – could start as user key storage and then change to system key storage
  - Storage could be freed and replaced with system key storage

How to do safely?
- Switch to user key temporarily
- Use MVCK, MVCSK, MVCDK, MVCOS to make a safe copy

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

How could this be exploited?

Unauthorized user can still cause system code to be interrupted

- Asynchronous abends (cancel, detach, etc.)
- Dispatcher interrupts
- Timer interrupts
  - Exits run on same task (via IRB) and can view status of system service and alter environment of task before returning control to system service
- TSO attention interrupts

How to do safely?
- Switch to user key temporarily
- Use MVCK, MVCSK, MVCDK, MVCOS to make a safe copy

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

VULNERABILITY PATTERNS ON Z/OS

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Vulnerability Patterns for z/OS

1. The Unintentionally Authorized PC
2. Untrusted Parms, Untrusted Regs
3. Untrusted, Indirectly Anchored Parms
4. Control Block Masquerade
5. Buffer Overflow
6. Index Out-of-Bounds

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

1 – The Unintentionally Authorized PC

Critical keyword on the ETDEF service defining a PC:

AKM

(The Authorization Key Mask)

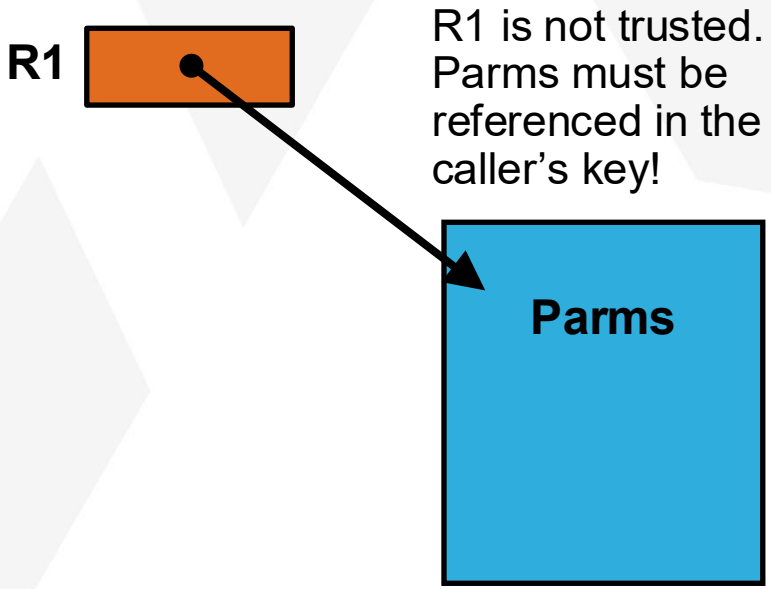
AKM(0) restricts the PC usage to callers running in key 0

AKM(0:15) allows the PC to be used by any caller

If a PC target routine is *intended for authorized callers* but ***inadvertently allows unauthorized ones***, it's highly likely to have an exposure!

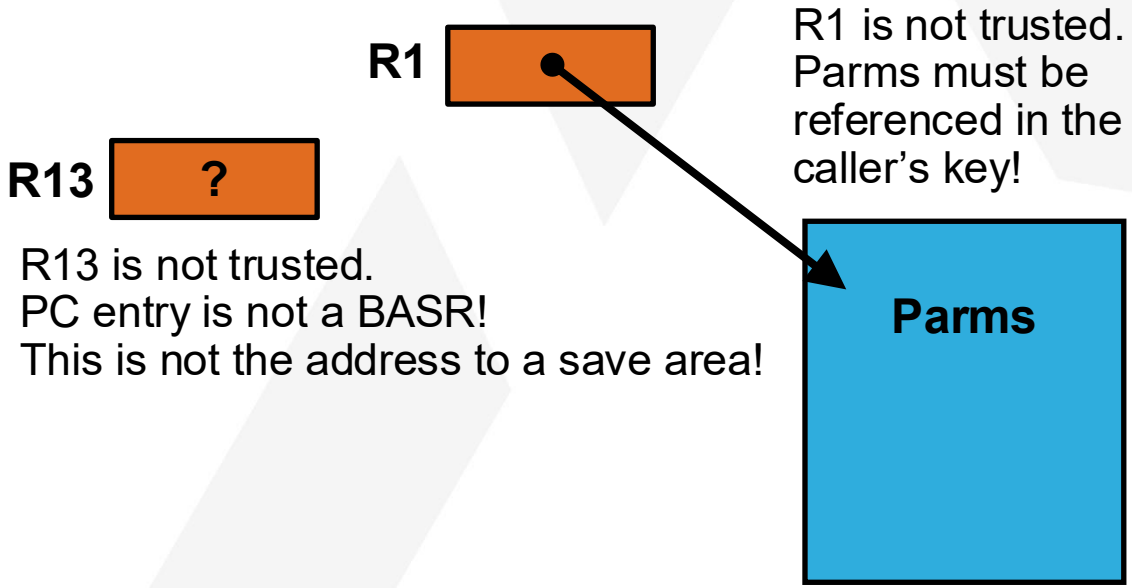
Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

2 – Untrusted Parms, Untrusted Regs



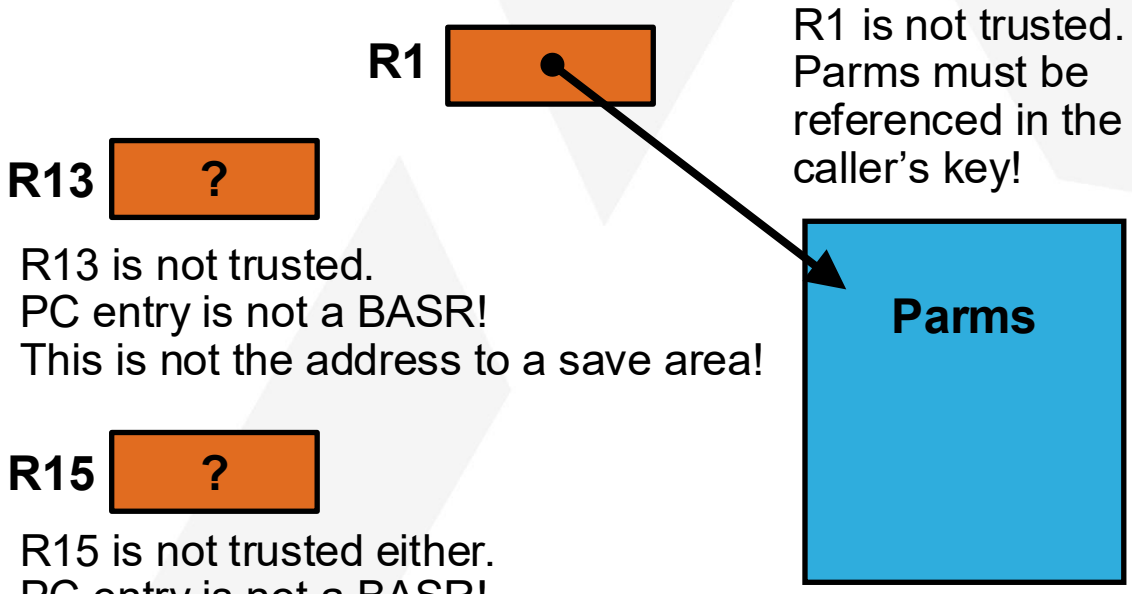
Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

2 – Untrusted Parms, Untrusted Regs



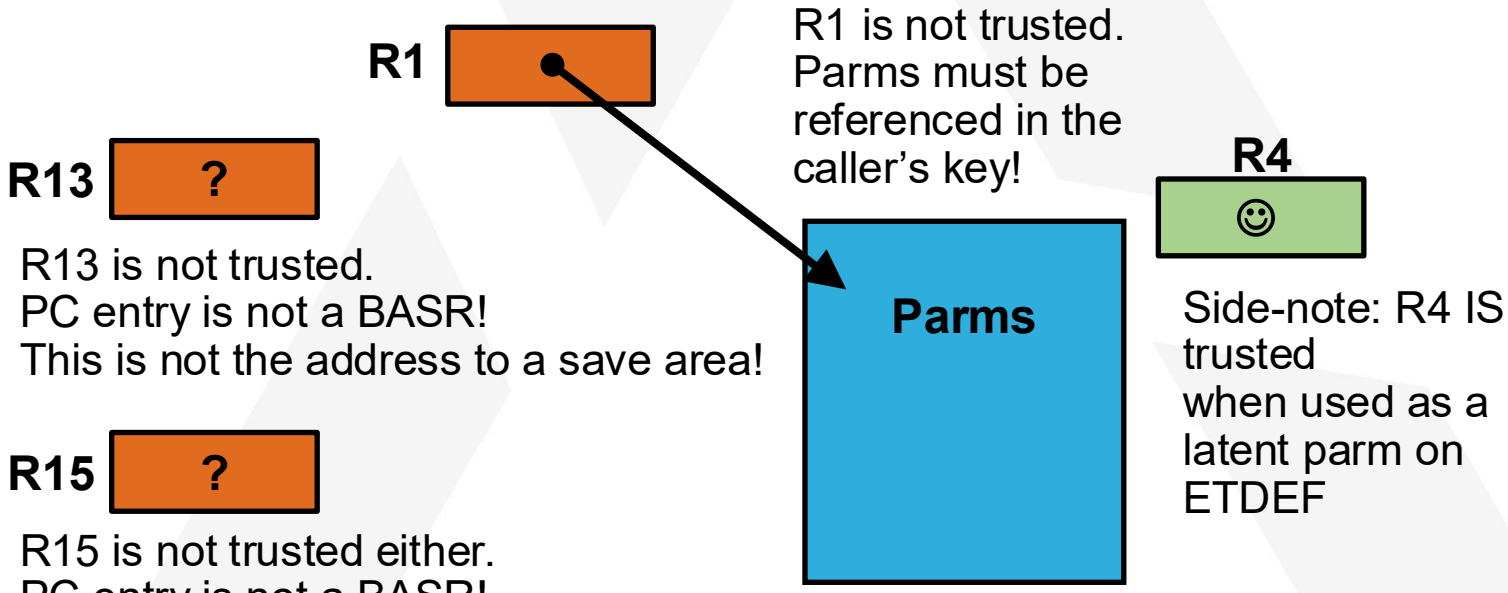
Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

2 – Untrusted Parms, Untrusted Regs



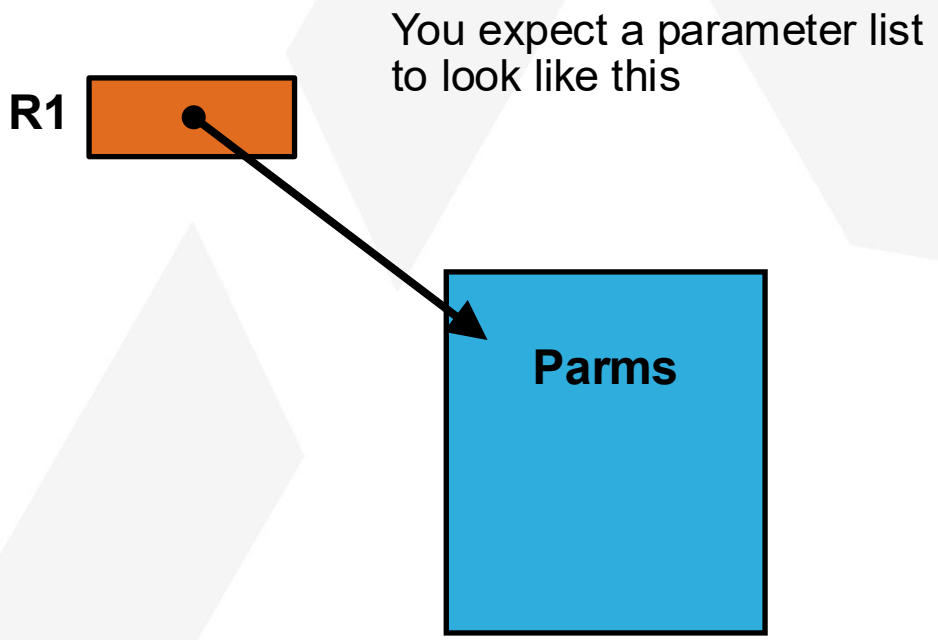
Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

2 – Untrusted Parms, Untrusted Regs



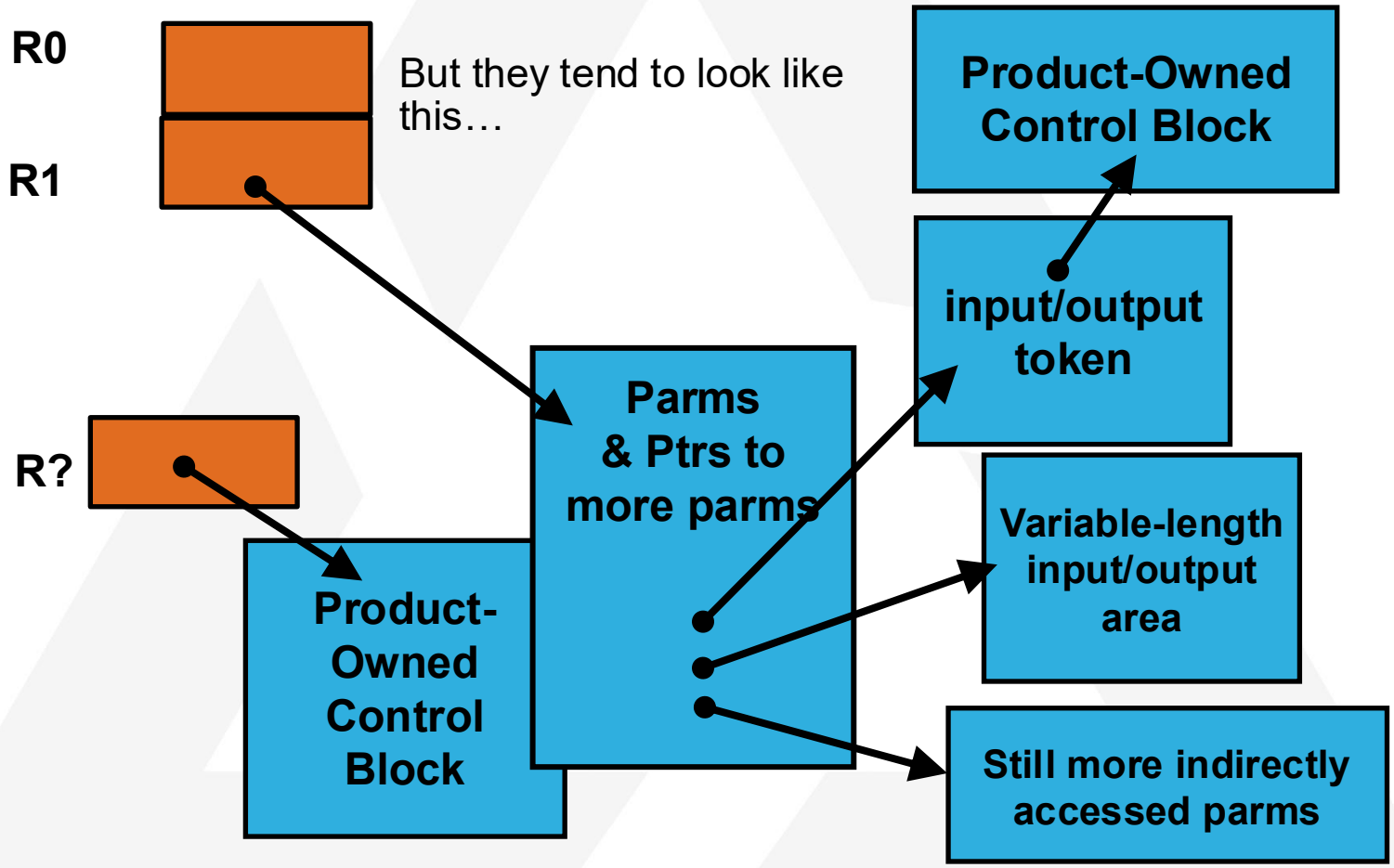
Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

3 – Untrusted, Indirectly Anchored Parms



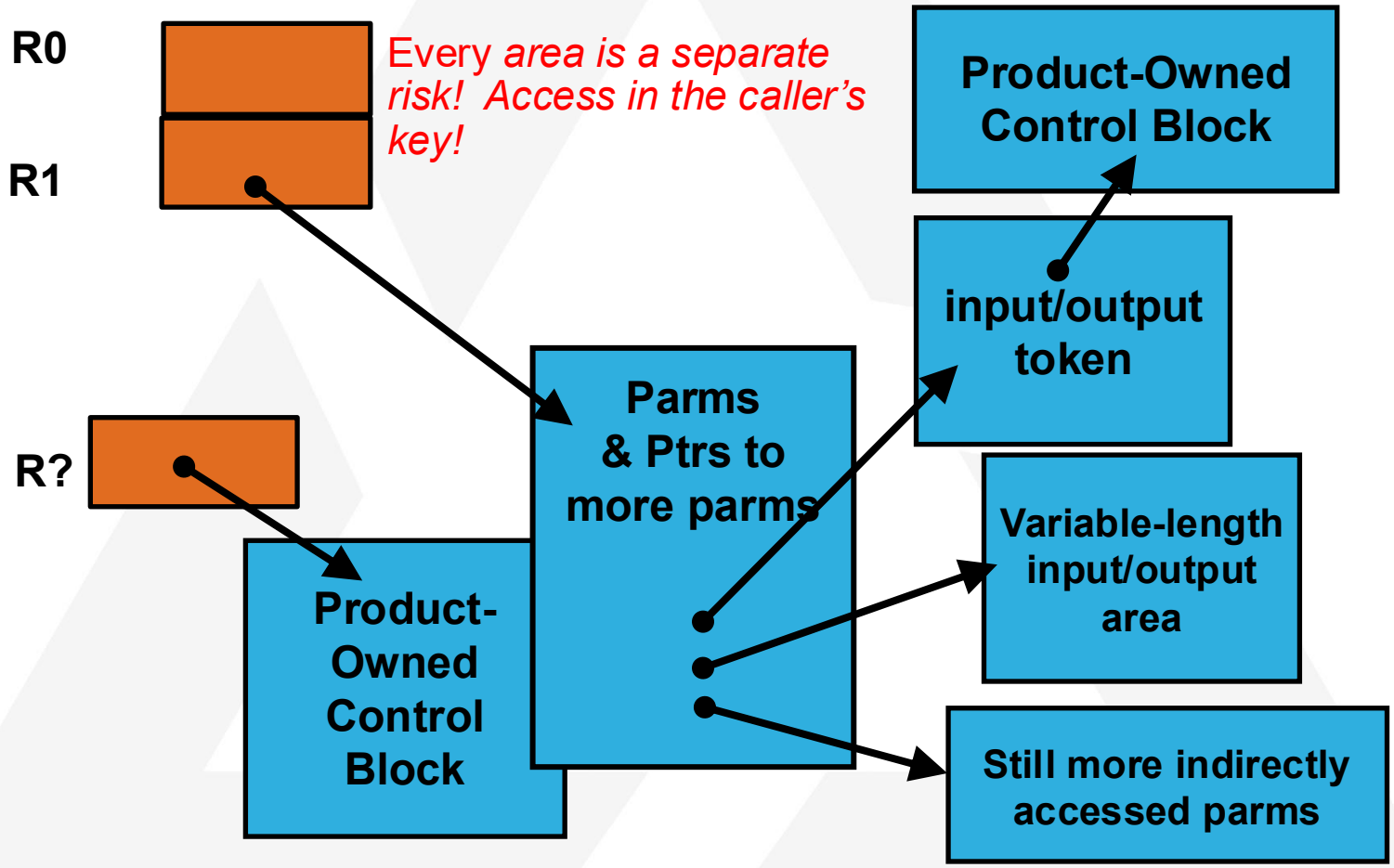
Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

3 – Untrusted, Indirectly Anchored Parms



Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

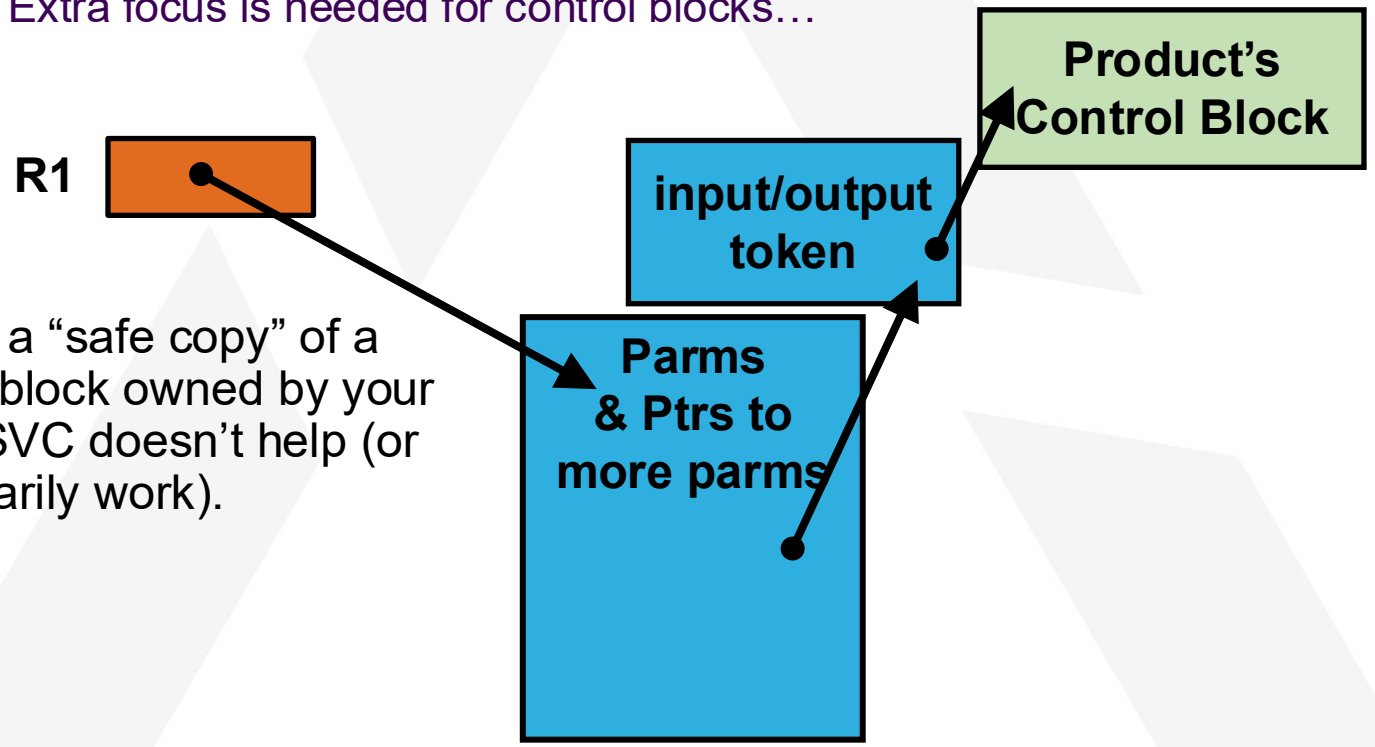
3 – Untrusted, Indirectly Anchored Parms



Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

4 – Control Block Masquerade

Extra focus is needed for control blocks…

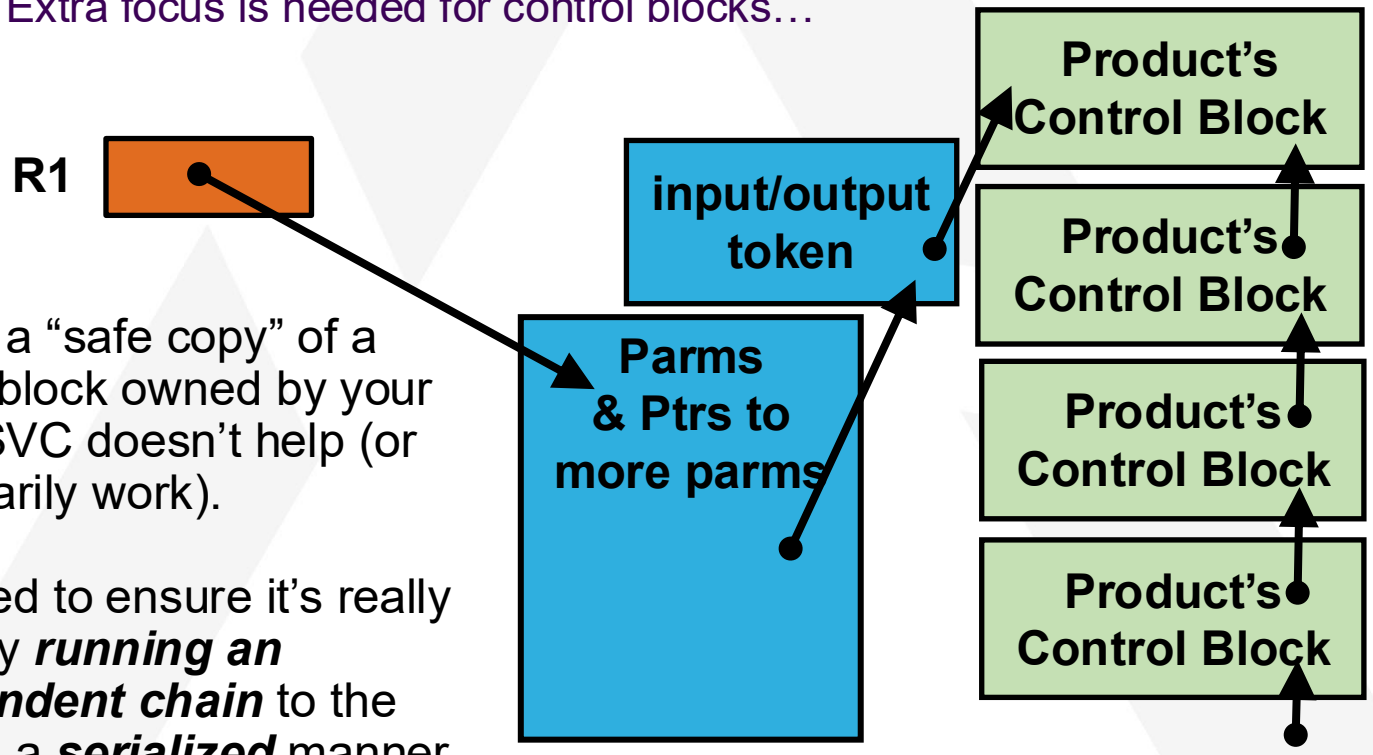


Making a "safe copy" of a control block owned by your PC or SVC doesn't help (or necessarily work).

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

4 – Control Block Masquerade

Extra focus is needed for control blocks…



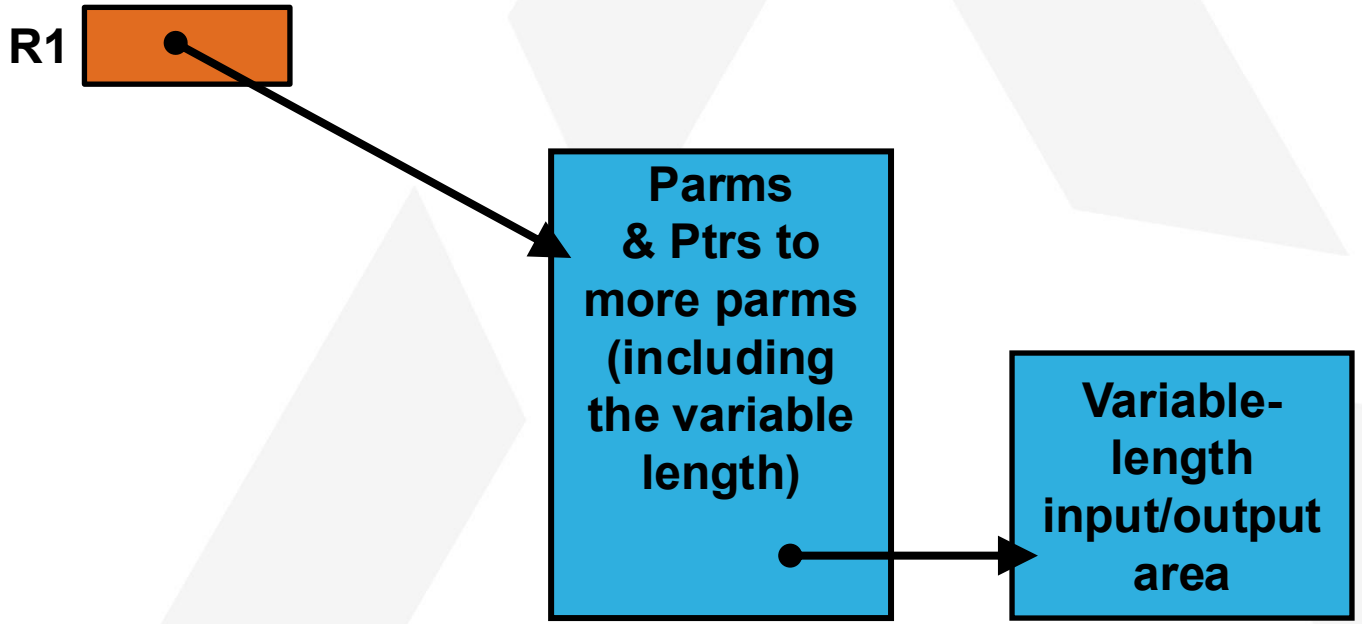
Making a "safe copy" of a control block owned by your PC or SVC doesn't help (or necessarily work).

You need to ensure it's really yours by ***running an independent chain*** to the block in a ***serialized*** manner.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

5 – Buffer Overflow

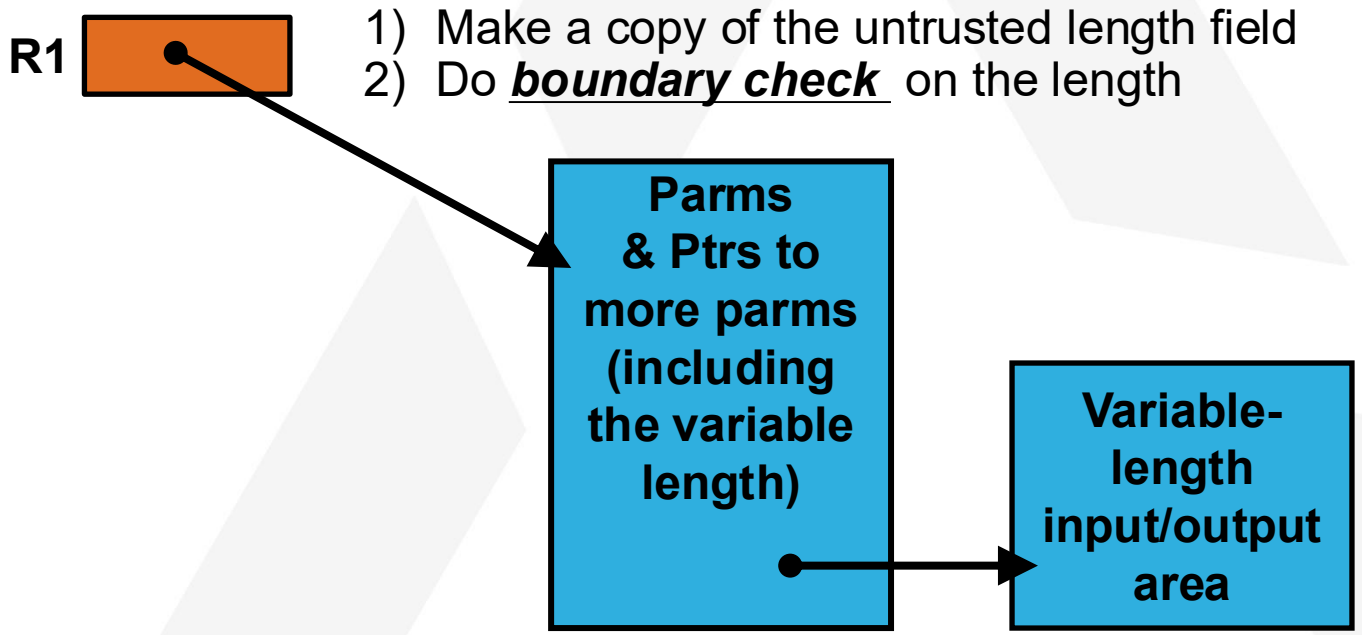
Extra focus is also needed for variable length areas



Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

5 – Buffer Overflow

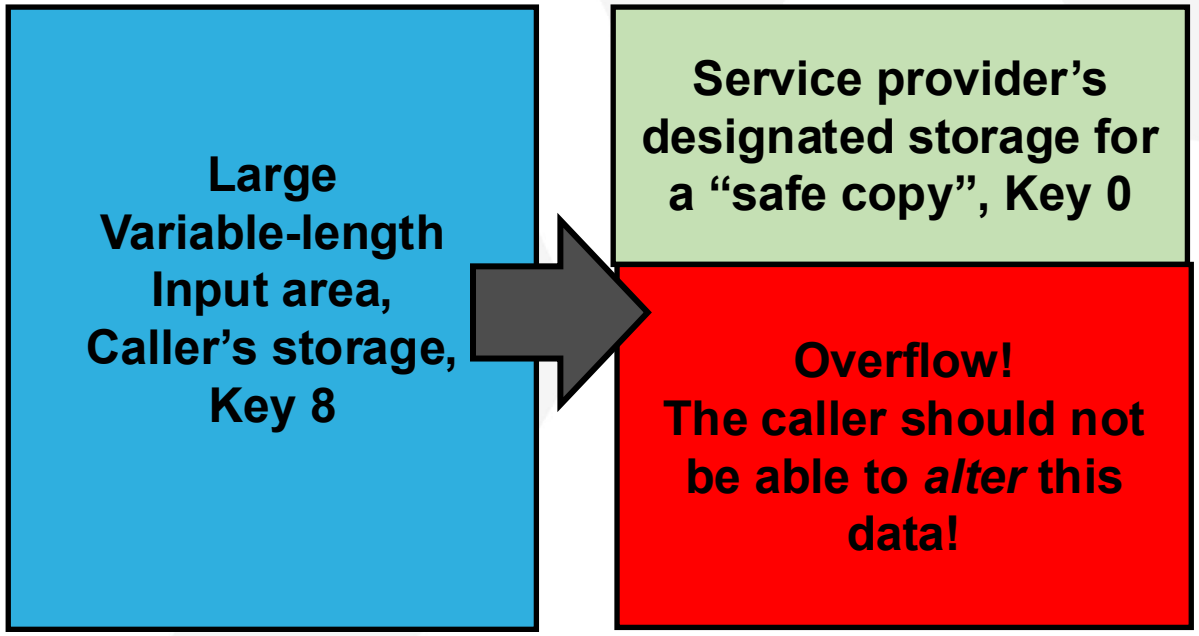
Extra focus is also needed for variable length areas



Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

5 – Buffer Overflow

Clarifying the overflow of target area when copying from the caller's input area



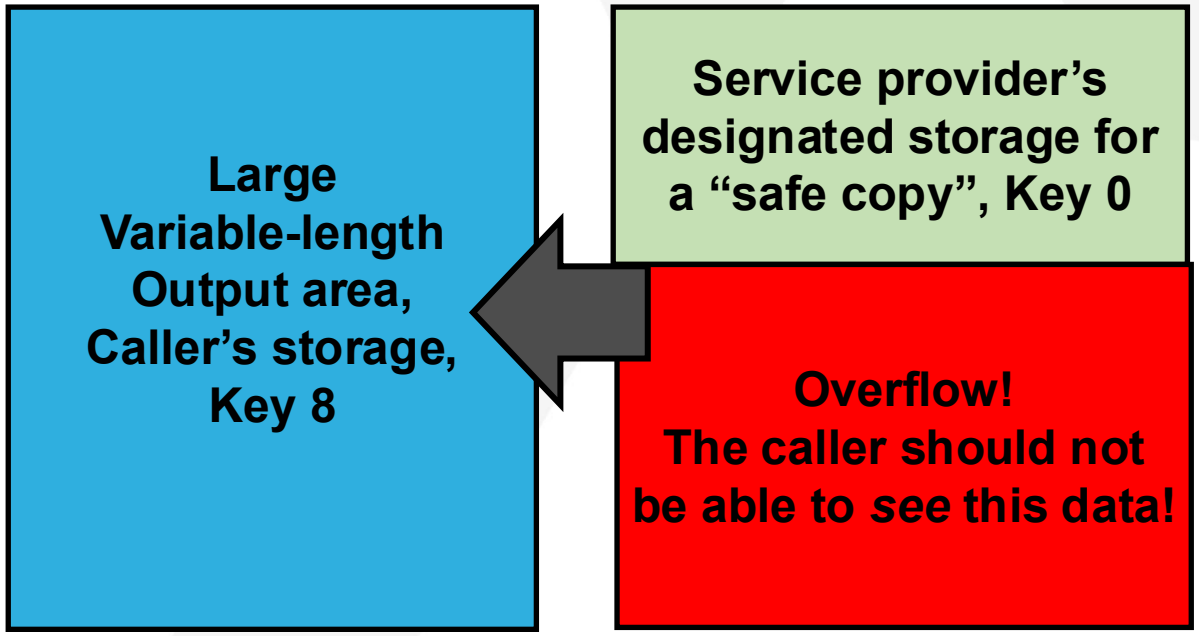
Copying with the caller's key does not protect against overflow!

Length boundary must be checked!

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

5 – Buffer Overflow

Clarifying the over-read from source area into the caller's output area



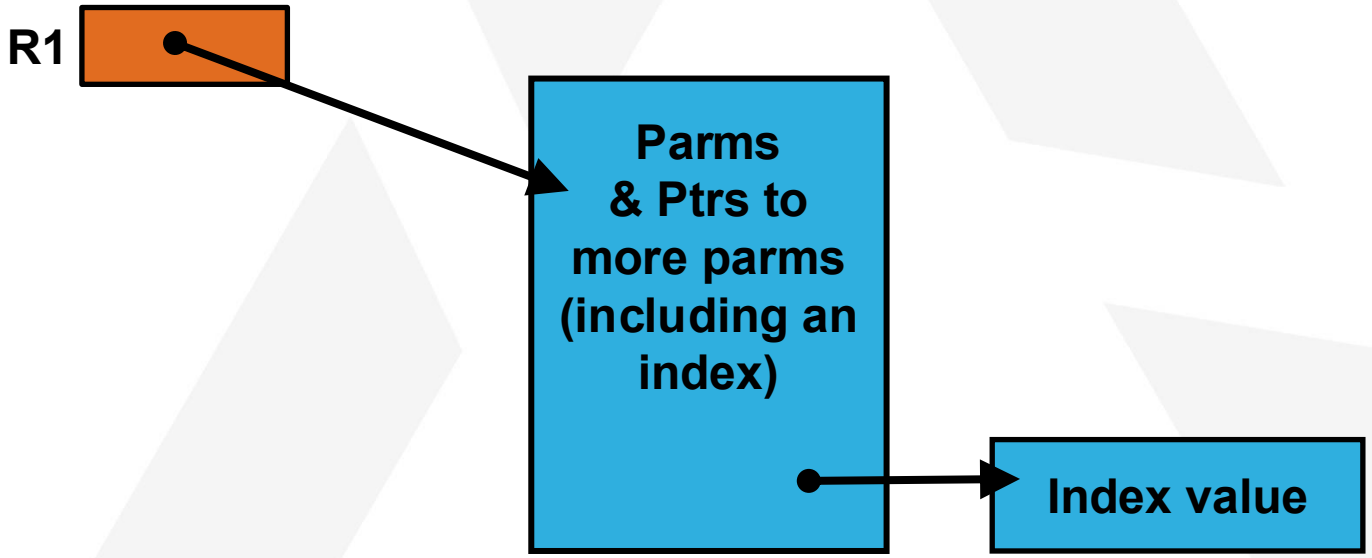
Copying with the caller's key does not protect against overflow!

Length boundary must be checked!

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

6 – Index Out of Bounds

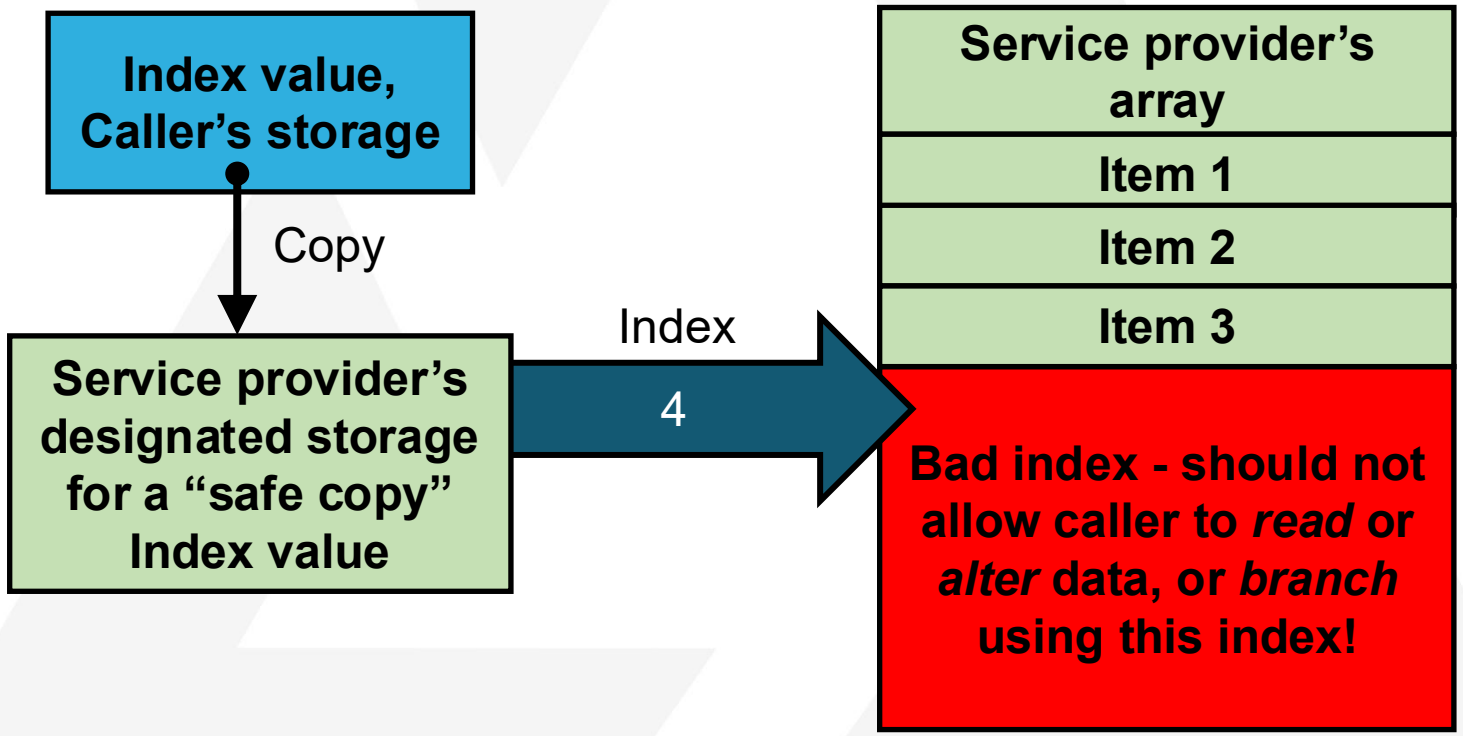
Extra focus is also needed for an index into a table or array



Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

6 – Index Out of Bounds

1) Make a copy of the untrusted index value using the key of the caller
2) Do ***boundary check*** . If index > 3 or index < 1, must reject.



Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

PROGRAM CHECK ANALYSIS

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Does this happen to you?

```
IEA995I SYMPTOM DUMP OUTPUT 119
SYSTEM COMPLETION CODE=0C4  REASON CODE=00000010
 TIME=17.20.39  SEQ=12691  CPU=0000  ASID=00D7
 PSW AT TIME OF ERROR  470C1000   800084EE  ILC 4  INTC 10
   ACTIVE MODULE          ADDRESS=00000000_00008000  OFFSET=000004EE
   NAME=MATTS
   DATA AT PSW  000084E8 - B5005870  A2205050  701C1F66
   AR/GR 0: 008FB37C/00000000_00000001   1: 00000000/00000000_EEEEEEEE
         2: FFFFFFFF/FFFFFFFF_0441BFF4   3: FFFFFFFF/FFFFFFFF_00000000
         4: FFFFFFFF/FFFFFFFF_008D3D40   5: FFFFFFFF/FFFFFFFF_044192B8
         6: FFFFFFFF/FFFFFFFF_00000003   7: FFFFFFFF/FFFFFFFF_EEEEEEEE
         8: FFFFFFFF/FFFFFFFF_044192B8   9: FFFFFFFF/FFFFFFFF_00000000
         A: FFFFFFFF/FFFFFFFF_00007198   B: FFFFFFFF/FFFFFFFF_80008000
         C: FFFFFFFF/FFFFFFFF_00008FFF   D: FFFFFFFF/00000000_00007198
         E: 00000000/00000000_800080E6   F: 20000000/00000000_00000000
 END OF SYMPTOM DUMP
```

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

That's a program check

- Program checks (especially in an authorized state) indicate all kinds of failures
  - Some are vulnerabilities
  - Some are just bugs
- Many types of program checks
  - z/OS Abend and Reason codes
  - Other useful classifications
- Main Question: **Could this happen again in a way that does not ABEND?**

Copyright© by SHARE Inc. Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Fetch Violations

- Abend 0C4's or 0E0's
  - Translation Exception Address (TEA) with x'00000800' bit active
- Is the program running in/target data in system key (0-7)?
  - Sometimes an authorized caller messes up too
- Could the task be triggered by a user via PC or SVC?
- Is there potential for data exfiltration:
  - Is the task interruptible? In an address space where a user could do so?
  - Could the data be loaded into some report or log?

Copyright© by SHARE Inc. Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Store Violations

- Abend 0C4's or 0E0's
  - Translation Exception Address (TEA) with x'00000400' bit active
- Is the program running in/target data in system key (0-7)?
  - Sometimes an authorized caller messes up too
- Is the target address consistent enough to always fail?
- Is this code accessible to unauthorized callers?
  - Could they instead trigger this indirectly?

Copyright© by SHARE Inc. Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Wild Branch

- Abend 0C6's or 0C1's
  - Can also be an Abend 0C4 on the instruction fetch
- Are we running in user key and problem state?
  - Do we have any other authorization (JSCBAUTH, AC(1))?
- Is the address consistent enough to always drive a failure?
- Can a user directly control the target address?

Copyright© by SHARE Inc. Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Denial of service

- Main Question: ~~Could this happen again in a way that does not ABEND?~~
  - **Does this abend cause system or service resiliency concerns**?
- Is the service availability an issue for all users?
- Does this impact or impair system performance?
- Who can trigger this behavior?
  - An average user?
  - Someone with some authority (Operator)?
  - An administrator of the service?

Copyright© by SHARE Inc. Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

WHAT NOT TO SAY WHEN WRITING CYBER RESILIENT SOFTWARE ON Z/OS

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

“I thought they checked the parameter list?”

PC routine A is available to unauthorized callers and PC routine B is only available to system key callers, but PC routine A passed a parameter from the user to PC routine B, causing PC routine B to overwrite storage in system key at the address from the user. This could still be a problem even if PC routine B is open to unauthorized callers if A runs in system key.

PC routine A was updated to obtain its own storage.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

“Why not just add the user input to the command?”

A network interface accepted a parameter with an identifier it wanted to add to a USS command, but it did not syntax check the input and used a syscall that allowed for multiple commands, so the user could append another command after the identifier which would be executed with UID 0 on the system.

Syntax checking was added and a safer syscall was used that only allows for one command to be run.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

“It’s in an APF library, so it must be safe to call.”

An authorized service allowed a user to specify any name for an exit which it would load and call, since only programs from APF libraries could be loaded, they assumed this was safe. The user specified a program that did not validate input and caused a buffer overflow giving the user’s program control.

The authorized service was updated to only use exit names if they match a system admin defined list.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

“Why would it matter if the module is reentrant?”

A PC routine open to unauthorized callers loaded a non-reentrant module and branched to it, key zero. The user set a stimer exit, overwrote the key eight code for the non-reentrant module after it loaded, and their instructions executed in PSW key zero.

The module was changed to be reentrant so that it would be loaded in key zero not user key storage.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

“We linked it as AC(1) just to be sure.”

Modules that did not expect to get control as a job step task with a parameter list were linked as AC(1), in one case because it was an alternate entry point to a load module and in another case, it was called by an AC(1) job step program. If invoked directly, both overwrote storage using system key zero.

Both entry points were changed to no longer be AC(1) because they were not really intended to be, although it was harder for the alternate entry point.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

“My SRB doesn’t need a purge TCB or ASID.”

An SRB routine that ran in another address space was not in private storage itself but relied on control blocks and data in the home address space, so when the scheduling address space was restarted the control blocks at those locations had changed but were still being used, leading to overlays.

The SRB was updated with a purge TCB and ASID to prevent the SRB from running after those terminate. Purge STOKEN is also available and recommended.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

“My ENF listener exit doesn’t need EOT or EOM.”

An ENF listener ran in its home address space and was loaded into private storage. It relied on control blocks and data in private storage and percolated to a recovery routine too, so when the home address space was terminated and replaced, the data and code at those locations changed but was still used.

The ENF exit was updated to add EOT and EOM yes, to stop from running after TCB or ASID termination.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

“If they used FORCE ARM, it’s their fault not ours.”

A started task was waiting on ECBs in storage that was being freed by their resource managers during address space termination. FORCE ARM led to their recovery routines getting control in unexpected ways and caused RTM to post and update ECBs in common storage that had been freed and reused.

Termination processing was updated to avoid freeing storage while it was still being used.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

“We get the storage, so no need to check the size.”

A space switching PC routine was obtaining storage in a system address space using a size specified by the user. This allowed a PC caller to occupy all the available storage in the system address space and prevent any other requests from being processed.

The PC routine was updated to check the size first.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

“Won’t the system initialize that to zero anyways?”

An authorized program forgot to initialize some registers and storage but had lucked out and the compiler set those registers to values that led to harmless abends and the storage was never used. After a recompile the register values led to storage overlays in one case and the storage was now used in another case, so residual data was now a pointer.

The program was updated to initialize the data and registers, not rely on the compiler or residual data.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

SHARE
EDUCATE • NETWORK • INFLUENCE

“We can’t get serialization due to performance.”

A monitor task was running a control block chain in a target address space from an SRB it scheduled there and writing out the data out in a report for its users. Since there was no serialization, unexpected system key data was being written out to the report.

Even if serialization might not have been practical, addition checks were added to verify the data again before anything was written to the report for users.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

“We need to display all that data for diagnostics.”

A recovery routine that received control in system key was running a save area chain based on the address in register 13 at the time of the error and displaying all the save area data for diagnostics, even if register 13 was not pointing to a save area.

The recovery routine was redesigned to stop displaying the data and just get a dump instead, unless the failing PSW was in a range that was safe.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

"This code is older than you are and no one complained about it. We should not have to change it."

A started task with a missing null pointer check was using data from low storage instead of the expected control block chain to find and update a control block. If zero it would usually abend and recover, but in some scenarios the data in low storage could point to data that it would overwrite using key zero.

The started task was updated to check for zero first.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

“It recovers from the abend, so it’s not vulnerable.”

After a module was extended to add a new base register the recovery routine was not updated to restore the new base register. If an abend occurred and it retried to a point where the base register was needed it would usually abend and recover from that abend as well, but in some cases, it would overwrite unintended storage in key zero due to the unexpected value found in the base register.

Recovery was updated to restore the register.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

“Isn’t everybody using Amode31 by now?”

A new exit was added but the authorized program specifying the address of the exit forgot to turn on the high order bit, which the service it was calling used to determine which Amode to call the exit in. As a result, it cut off the high order byte and called an address below the line where a user could place their own program to get control key zero, instead.

The high order bit was turned on to fix this.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

“The POST to the missing ASID should safely ABEND”

A POST specifying an ASCB was done to an ASID that could terminate. When the ASID terminated the storage for the ECB was being freed and reused causing POST to overwrite unintended storage in the new ASID.

The Safe XM Post service IEAMSXMP was used to avoid this.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

SHARE
EDUCATE • NETWORK • INFLUENCE

“We need to free that code to avoid a common storage leak”

It is true that obtaining common storage every time a started task or other service is started could lead to a storage leak. However, if other address spaces could be executing that code in an authorized state, which is usually possible for modules in common storage, freeing the storage out from under them can cause unintended code to execute.

The storage was anchored to a persistent control block or in one case an authorized name token from the name token service and reused.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

“No one should be calling this service that way”

A product used a combination of PC routines (available to unauthorized callers) to route calls to individual services based on the passed function code. There was some boundary checking, but some services could be invoked through “unintended” paths, driving them to use uninitialized data.

The PC routines were amended to prevent these “unintended” paths.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

“This product doesn’t need to recover from this sort of ABEND”

A product was designed with minimal recovery for ABENDs experienced by any of the libraries that it called. The justification was that if these libraries encountered such ABENDs that no data in the operating environment could be trusted and the service should terminate. The problem with this approach is that it escalated small-scale issues with single threads to problems that rendered the product inoperable until manual intervention restored it.

The product has been changed to recover gracefully, and selectively terminate untrusted threads.

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

REFERENCE MATERIAL

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Important Links

- z/OS System Integrity Statement
  - https://www.ibm.com/downloads/cas/OWGOKG40
- IBM Z Security Portal FAQ
  - https://www.ibm.com/downloads/cas/EAO940BR
- z/OS MVS Programming: Authorized Assembler Services Guide
  - https://www.ibm.com/docs/en/zos/3.1.0?topic=guide-protecting-system

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/

Experience more with IBM

Visit us at the IBM Booth #113

After a full day of technical sessions, take a break with us!

Connect with our experts, snap a photo with the z17 Plexi or the latest Telum II, and get an up-close look at our Spyre Accelerator.

Come back each day for fresh topics and demos at our expert stations.

Think 2026

Join 5000+ senior business and technology leaders who are seizing the AI revolution to unlock unprecedented growth and productivity at Think 2026.

Find out more information using the QR code below.

IBM Digital Asset Haven

IBM Digital Asset Haven is the operational backbone for financial institutions and regulated enterprises entering the digital asset economy.

Find out more information using the QR code below.

IBM Corporation 2026

Your feedback is important!

SHARE
EDUCATE • NETWORK • INFLUENCE

Submit a session evaluation for each session you attend:

www.share.org/evaluation

Copyright© by SHARE Association Except where otherwise noted, this work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivs 3.0 license. http://creativecommons.org/licenses/by-nc-nd/3.0/