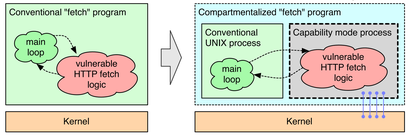
CTSRD

Peter G. Neumann, Robert N. M. Watson, and **Simon W. Moore**
Jonathan Anderson, Ross Anderson, David Chisnall, Nirav Dave, Brooks Davis, Rance DeLong, Khilan Gudka, Steven Hand, Alex Horsman, Jong Hun Han, Asif Khan, Myron King, Ben Laurie, Patrick Lincoln, Anil Madhavapeddy, Ilias Marinos, Dr Theo A. Markettos, Ed Maste, Andrew W. Moore, Alan Mujumdar, Prashanth Mundkur, Steven J. Murdoch, Robert Norton, Philip Paeps, Michael Roe, Colin Rothwell, John Rushby, Hassen Saidi, Muhammad Shahbaz, Stacey Son, Richard Uhler, Philip Withnall, Jonathan Woodruff, Bjoern A. Zeeb

Ed Maste, Prashanth Mundkur, Steven Murdoch, Jong Hun Han, Hassen Saidi, Khilan Gudka, Colin Rothwell, Peter G. Neumann, Malte Schwarzkopf, Brooks Davis, Nirav Dave, Jonathan Woodruff, Bjoern Zeeb, Robert Norton, David Chisnall, Alan Mujumdar, Alex Horsman, Andrew Moore, Simon Moore, Robert Watson

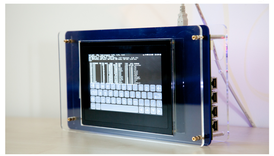
Capsicum

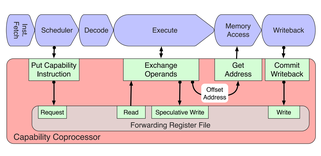
Capsicum is an OS-based **hybrid capability system** that supports **application compartmentalization** to mitigate security vulnerabilities. Capsicum shipped in FreeBSD 10, with Google-developed patches available for Linux.



Capability systems are designed to implement the **principle of least privilege**. This mitigates both **known and unknown vulnerabilities and attack techniques**. However, current **CPU architectures scale poorly** when isolating multipart programs, and **provide poor programmability**.

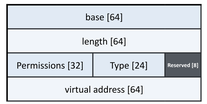
CHERI: Capability Hardware Enhanced RISC Instructions





Hardware-Based Memory Capability Model

CHERI supports scalable, in-process, compiler-directed, fine-grained memory protection. CHERI combines the security of a **tagged memory capability model** with the C-language friendliness of **hardware-assisted fat pointers**. This mitigates memory-based exploit techniques such as buffer overflows, return-oriented programming. CHERI is suitable for use in C-language TCBs (such as our adapted CheriBSD OS) and also higher-level languages (such as Java or OCaml). CHERI scales better than conventional Memory-Management Unit (MMU) techniques or SFI techniques.

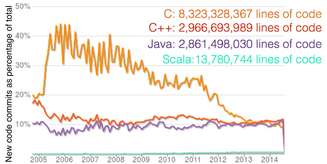


CHERIv3 capabilities can be **used as C pointers.**

Hardware protection from buffer overflows in 'unsafe' languages

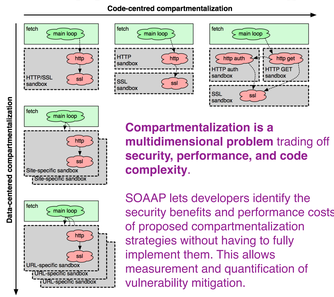
```
void initBuffer(void) {
  // Allocate on-stack buffer
  int buffer[21];
  // Pass a pointer to another f
  overflow(buffer);
}
void overflow(int *x){
  // Write over the end of the b
  x[42] = 12;
}
```

```
# In initBuffer():
# Get the buffer's offset in the current stack frame
daddiu  $3, $fp, 4
# Set $2 to the buffer size
daddiu  $2, $zero, 84
# Derive a capability to the buffer from the stack
# capability in the first capability argument register
cincbase  $c1, $c11, $3
csetlen $c3, $c1, $2
# In overflow():
# Store 12 past the end of the buffer:
daddiu  $1, $zero, 168
daddiu  $2, $zero, 12
csw  $2, $1, 0($c3)   # This will trap at run time
```



Unsafe languages are not going away!

SOAAP: Security-Oriented Analysis of Application Programs



Compartmentalization is a multidimensional problem trading off **security, performance, and code complexity**.

SOAAP lets developers identify the security benefits and performance costs of proposed compartmentalization strategies without having to fully implement them. This allows measurement and quantification of vulnerability mitigation.

Annotate with past vulnerabilities

```
1  __soaap_sandbox_ephemeral("parser")
2  void parse(__soaap_read_fd int ifd, DOMTree* t) {
3    ...
4    if (...) {
5      __soaap_vuln_pt("CVE-2005-ABC");
...
13 }
14
15 __soaap_vuln_fn("CVE-2005-DEF")
16 void not_sandboxed() {
...
18 }
```

```
$ make soaap
** Sandboxed method "parse" has a past-vulnerability
** annotation for "CVE-2005-ABC". A new vulnerability
** here would leak the following:
++ Read access to file descriptor "ifd"

** Method "not_sandboxed" had past vulnerability
** "CVE-2005-DEF" but is not sandboxed. Another
** vulnerability here could leak ambient authority
** including full network and file system access.
```

Reports show the rights an attacker exploiting the vulnerability would gain with the current compartmentalization strategy.

CHERI Clang/LLVM Support

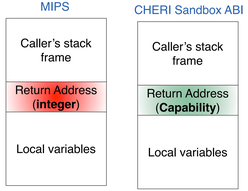
Incremental adoption requires **backwards compatibility,** so our **LLVM**-based compiler supports **three ABIs** with CHERI.

More compatible ⟷ More safe

MIPS n64
Pure MIPS
Pointers are integers

n64+CHERI
Compatible with n64
Annotated pointers are capabilities

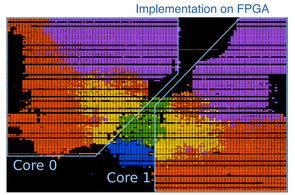
Sandbox
All pointers are capabilities



Old MIPS world: Overwriting the return address on the stack with MIPS and other conventional ISAs allows **ROP exploits.**

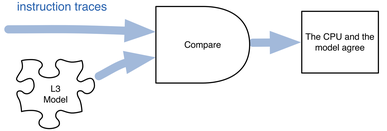
New CHERI world: Overwriting the return address on the stack with CHERI causes a tag failure and **run-time trap.**

CHERI Processor and ISA Testing and Verification



Full system instruction traces

L3 formal model of CHERI is the **first formal model** to **boot a full commodity OS.**



Members of the CTSRD team and its external oversight group at our May 2011 review meeting in Cambridge, UK

Joe Stoy (Bluespec), Jonathan Woodruff (Cambridge), Ben Laurie (Google), Ross Anderson (Cambridge), Virgil Gligor (CMU), Philip Paeps (Cambridge), Li Gong (Mozilla), Peter Neumann (SRI)

Simon Cooper, Michael Roe (Cambridge), Robert Watson (Cambridge), Howie Shrobe (DARPA), Steven Murdoch (Cambridge), Sam Weber (NSF), Jonathan Anderson (Cambridge), Simon Moore (Cambridge)

Anil Madhavapeddy (Cambridge), Dan Adams (DARPA), Rance DeLong (LynuxWorks), Jeremy Epstein (SRI), Hassen Saidi (SRI)

Architectural extraction transforms the complex pipelined implementation into a much simpler instruction-set architecture model to be checked.

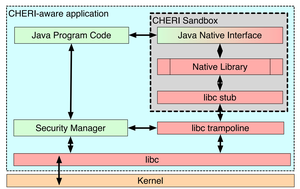
Hardware-Assisted, Object-Capability-Based Compartmentalization

Process-based sandboxing on current CPU architectures **does not scale to the tens or hundreds of thousands of compartments** required for contemporary applications such as web browsers or office suites. CHERI layers a **hardware-software object-capability model over the in-process capability memory model**. This allows efficient representation of both **asymmetric and mutual distrust** between application components.

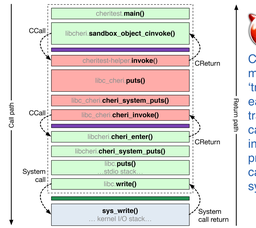
CheriBSD extends the open-source FreeBSD operating system with support for granular in-process memory protection and compartmentalization. Demo applications sandbox components such as packet processing and image rendering.

```
# tcpdump -i atse0 | head -20
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on atse0, link-type EN10MB (Ethernet), capture size 65535 bytes
02:21:41.658068 IP cheritest.sec.cl.cam.ac.uk.ssh > c0188.ow.cl.cam.ac.uk.49225: Flags [P.], seq 101973773:101973889, ack 1807744729,
win 1040, options [nop,nop,TS val 1123802448 ecr 91588452], length 116
02:21:41.871254 IP gw-2456.route-nwest.net.private.cam.ac.uk.1985 > 224.0.0.102.1985: HSRPv1
02:21:41.929941 STP 802.1w, Rapid STP, Flags [Learn, Forward], bridge-id 83e7.00:1c:0e:50:40:00.800f, length 42
02:21:41.946293 STP 802.1w, Rapid STP, Flags [Learn, Forward], bridge-id 806c.00:1c:0e:50:40:00.800f, length 42
```

Capsicum isolates tcpdump network processing in a single sandbox, which could allow successful exploits to gain network-sniffing access or watch/influence processing of other flows/packets. CHERI tcpdump enables **per-flow, per-packet sandboxing** plus **bounds checks** on all packet-buffer accesses.



Prototype CHERI-Java sandboxes native code using capabilities. Pointer errors in C can't affect Java memory; native I/O must be authorized by the Java security manager.



CheriBSD maintains a 'trusted stack' for each user thread, tracking object-capability invocations. Only privileged objects can invoke system calls.

TESLA: Temporally Enhanced Security Logic Assertions

Security properties are often **temporal.**

- Did something correctly lock this resource **before** using it?
- Will audit logs **eventually** be written to describe this event?
- Has an access control check **previously** been performed?

```
void use_cert(X509 *cert) {
#ifdef TESLA
  TESLA_WITHIN(main, previously(
    X509_STORE_CTX_init(ANY(ptr), ANY(ptr),
    cert, ANY(ptr)) == 1, X509_verify_cert(
    ANY(ptr)) == 1));
#endif
  /* use the certificate ... */
}
```

C-language source code annotated with temporal security expressions.

Generated automata describe allowed event sequences

```
automaton {
  identifier {
    location {
      filename: "openssl-api.c"
      line: 188
      counter: 0
    }
  }
  context: ThreadLocal
  expression {
    type: SEQUENCE
    sequence {
      expression {
        type: FUNCTION
        function {
          function {
            name:
              "X509_STORE_CTX_init"
          }
        }
...
```

Visualization helps developers understand program structure and identify security-policy failures (e.g., access without a corresponding check)

Run-time errors if constraints are violated

```
TESLA failure:
In automaton 'openssl-api.c:211#0':
automaton 0 {
  state 0: --(main(X,X): Entry)-->{1 <<init>>}
  state 1: --(X509_STORE_CTX_init(X,X,cert,X)==1)-->{2}
  state 2: --(X509_verify_cert(X)==1)-->{3}
  state 3: --(NOW)-->{4}
  state 4: --(main(X,X)==X)-->{5 <<cleanup>>}
  state 5:
}
openssl-api: No instance matched key '0x1
  [ 7fda614147c0 X X X ]' for transition(s)
  [ {3:0x1 -> 4} ]
```

Approved for public release. This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-10-C-0237. The views, opinions, and/or findings contained in this article/ presentation are those of the author/presenter and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense.