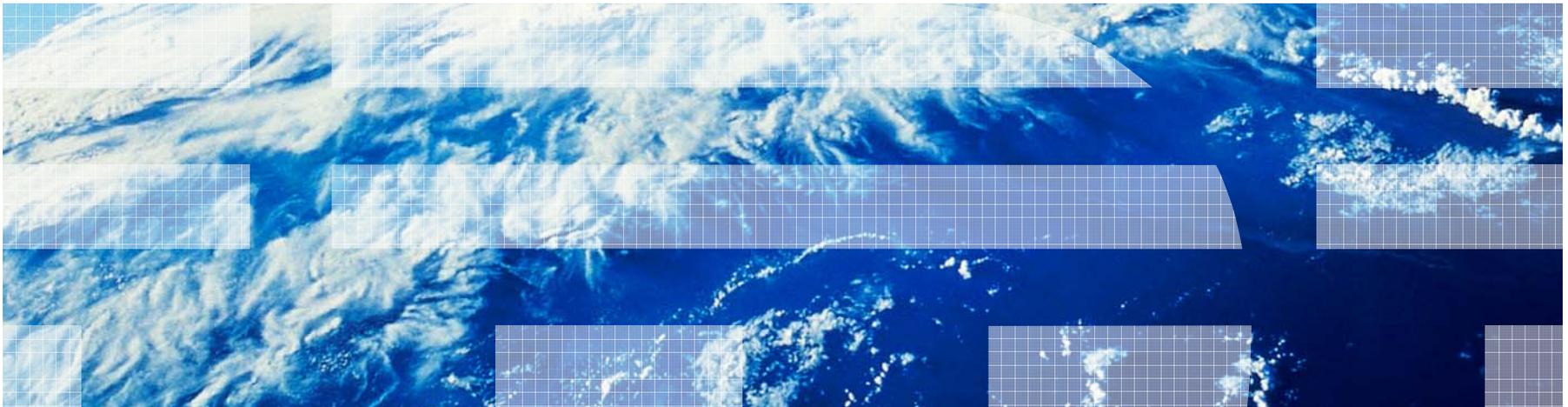


HWMAC: Hardware-Enforced Fine-Grained Policy-Driven Security

W. Eric Hall, Guerne D. H. Hunt, Paul A. Karger, Mark F. Mergen,
David R. Safford, and David C. Toll



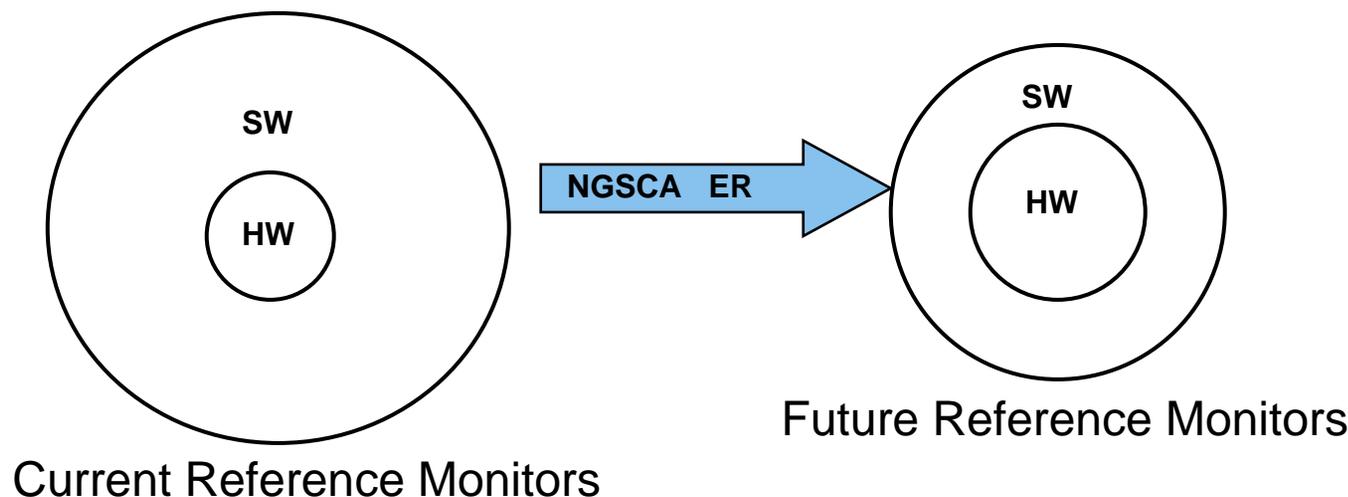
Next Generation Secure Processor Architecture

- IBM Research has been exploring a major paradigm shift in hardware security architecture
- Goal is to make building high security systems easier
- CPU security features originated in the late 1960s / early 1970s
 - Virtual memory with protection bits in page and/or segment descriptors
 - Protection rings or Capabilities
- Since then, semiconductor technology has changed drastically, but security features have not exploited those improvements
 - For the most part – we do have crypto hardware and co-processors
- Now we have massive multi-cores on a single chip
 - Software developers are having a very hard time making effective use of massive multi-core designs
- We could use some of the newly available transistors for new security features
 - Might have fewer cores, but might achieve much higher security at lower performance impacts
 - Might make achieving high assurance easier

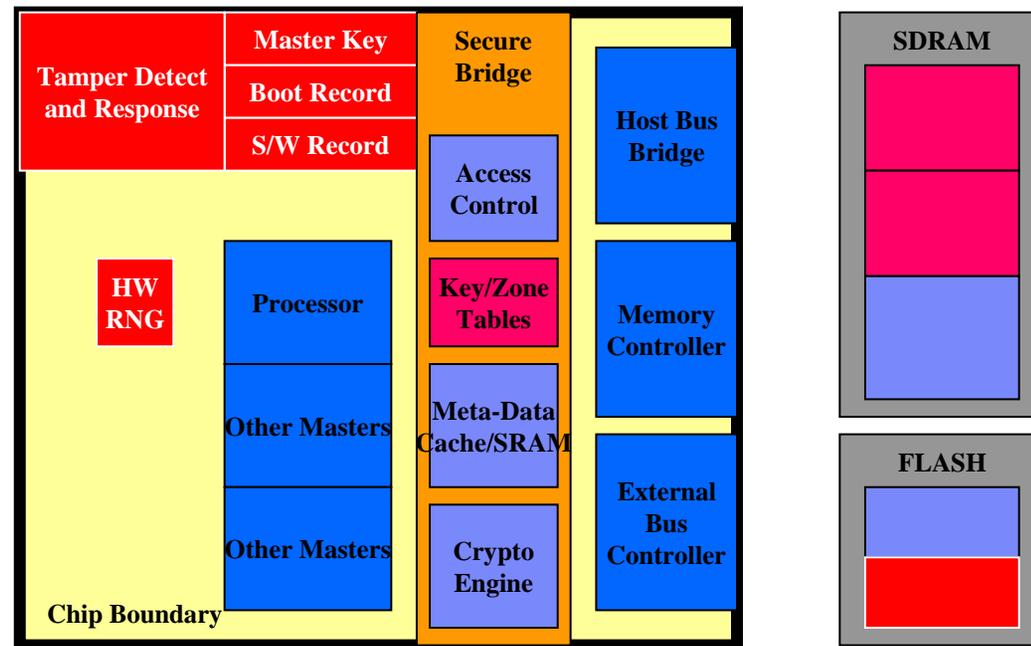
Project Objective

- The objective of this project is to develop and evaluate a computer architecture which enhances the security and isolation of all levels of software running in the resulting systems.
 - Ideally this new architecture should be instruction set neutral (except for new features).
 - Ideally it should be transparent to applications, although there will be some impact to operating systems and hypervisors.
 - Ideally it should provide better support for software as a service and service oriented architectures.

- Approach:
 - Reduce size and complexity of software portion of the Reference Monitor (TCB) by moving functions from software to hardware



Background



- IBM's existing Secure Processor Architecture teaches that hardware can make a substantial difference in security – essentially an approach to encrypt all of memory on an arbitrary chip design
- Covers one hardware threat model, and software attacks dependent on that vector, extremely well. However it does not address hardware and software attacks that it was not designed for
- New project addresses attacks/issues not addressed by existing Secure Processor Architecture and, for some technologies, builds upon that architecture
- A derivative of this architecture has been extensively used by IBM customers

Technologies Proposed by the ER

- There are nine technologies in different stages of investigation
 - Object and subject labelling with mandatory access control (**HWMAC**)
 - Tagged architecture with automatic state save
 - Architecture Support for Modular Software
 - Secure recursive virtualization
 - Logical partition memory:
 - a) Improved MMU with hypervisor translation
 - b) Recursive logical partition memory
 - c) Hardware supported recursive virtualization with hierarchical TLB.
 - Secure Message Passing Bus
 - Hardware enforced protection against timing channels
 - LPAR Isolation
 - Hardware support for modularizing the kernel

- Additional contributors: Rick Bovie, Tom Fox, Suzanne McIntosh, and Marcel Rosu
- These technologies are in various states of development, from proposed approaches to implemented and running in a simulator
- This talk focuses only on **HWMAC**
 - Only technology for which we have received export control clearance

Hardware-Enforced Fine-Grained Policy-Driven Security (HWMAC)

- Design motivated by traditional capability machines and the Linux SMACK, but with significant changes
- Labelling data in memory/cache/registers
 - The novel concept is to label all data/state in memory, cache, and registers, and provide full, policy based, instruction level, Mandatory Access Control in hardware
 - Only a small part of hypervisor needs to be trusted to manage labels
 - Errors in other software cannot result in secrecy or integrity data leaks
 - Policy can implement **MAC**, **capability**, **taint flow**, **injection attacks**, and **other security architectures including HW type checking**.



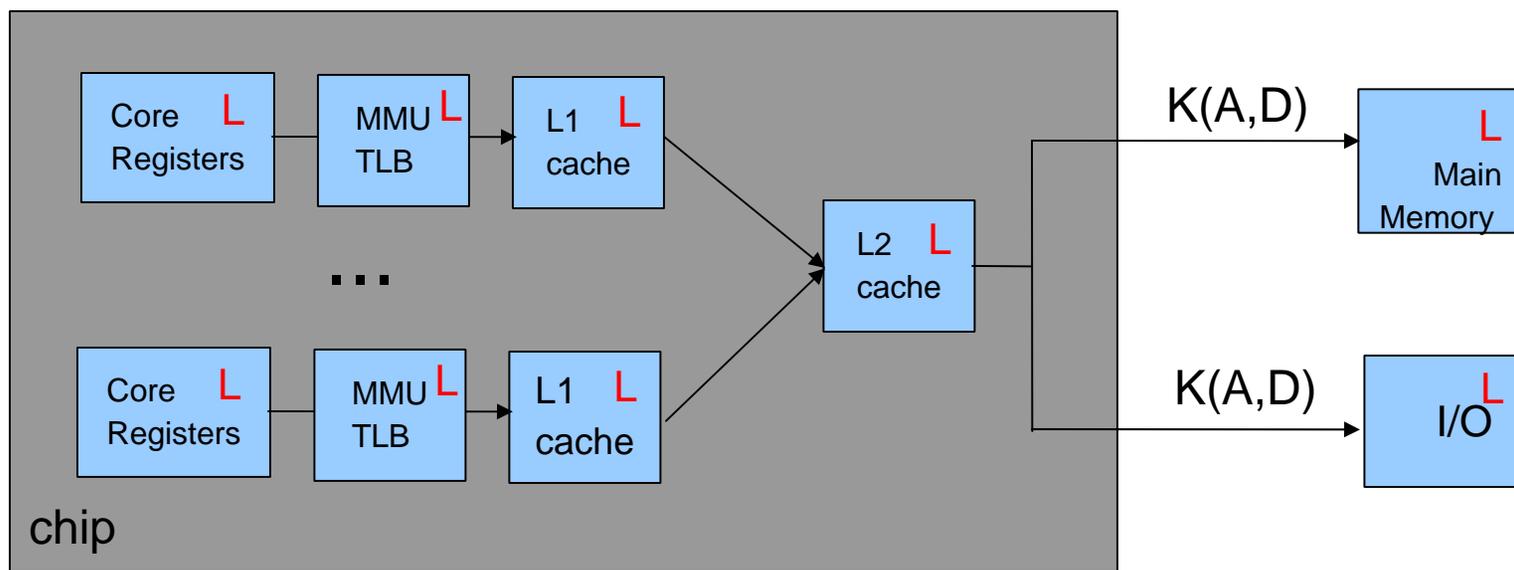
Demonstrated



Possibility

- We have focused first on **taint flow**, and **injection attacks** as they have the most commercial relevance

Labelling Concept



- Labelled memory with mandatory access control (new) L
- The data is labelled wherever it exist in the system

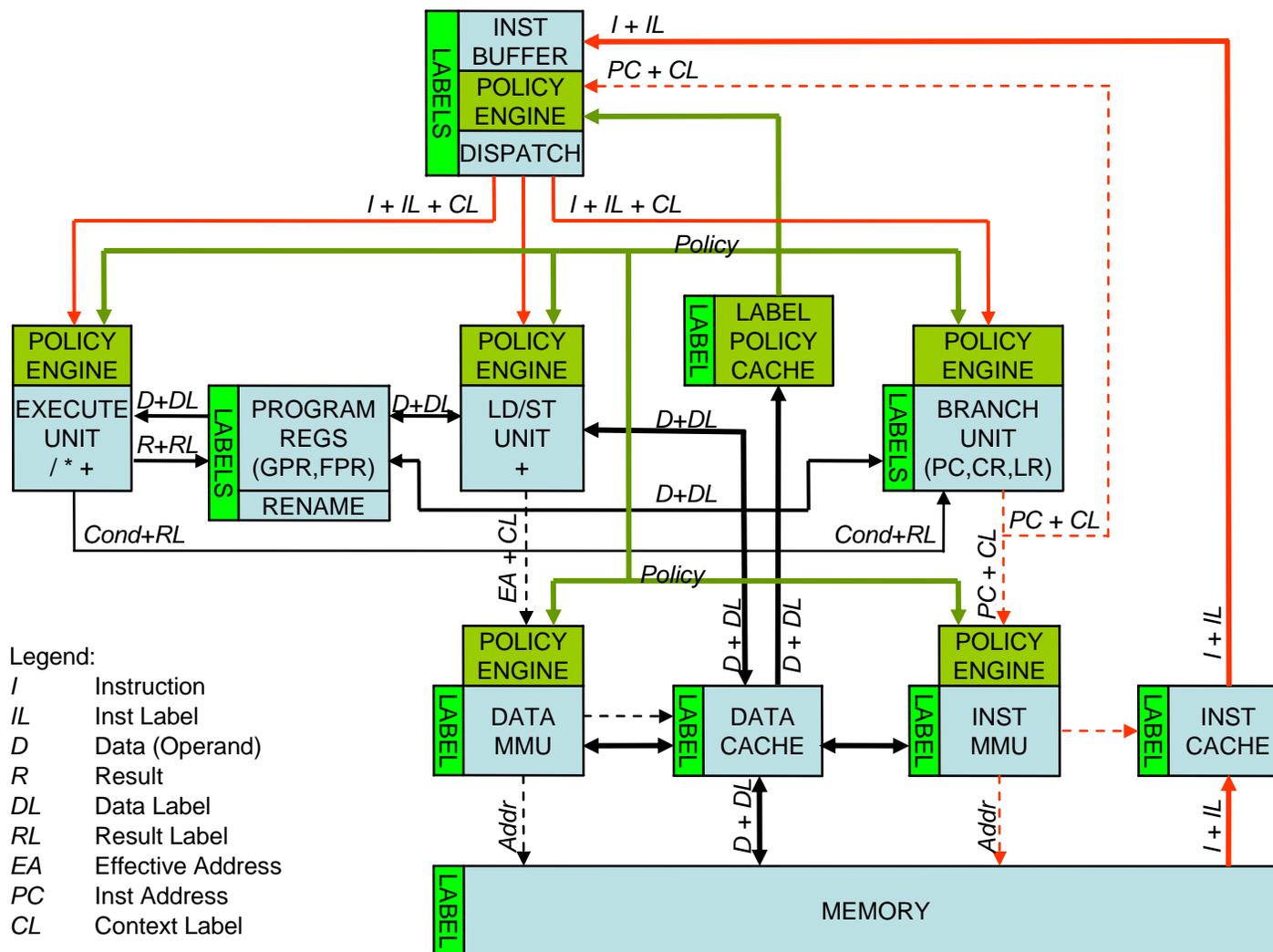
Hardware Mandatory Access Control Policies

- Theoretical Basis for byte label sanitization
 - Z. Su and G. Wassermann, “The Essence of Command Injection Attacks in Web Applications”, POPL '06
 - proof that byte level labeling can support “perfect” sanitization for any CFG (SQL, shell, XSS) under reasonable assumptions.
- Smack (Linux kernel MAC) provides Lampson Access Matrix

Subject/Object	1	...	N
1	RWX	RX	*
...	RX	RWX	W
N	*	W	RWX

- Implementations typically simplify this
 - Capabilities (one row at a time)
 - Access Control List (one column at a time)
 - Separate integrity and secrecy tables
- Policies must be defined for each CPU instruction and can be dynamically loaded into the CPU policy engines

Impact of HWMAC on Processor Architecture



HWMAC for traditional MLS

- The most obvious use of hardware MAC labels is to implement traditional mandatory policies, such as Bell and Lapadula secrecy or Biba integrity
- The potential payoff for such an implementation is to provide much more fine grained control of the data than simply at the granularity of files
- Historically, there have been two major software subsystems built that required finer granularity
 - Multics message segments for MLS email – each message carried its own label
 - Required significant increase in the size of the TCB
 - Multi-level secure database management systems (MLS DBMS)
 - To avoid having to trust the entire MLS DBMS, the Hinke-Schaefer and Seaview DBMSs partitioned the data by storing information for each secrecy access class in a different file or segment
 - Untrusted DBMS processes could see only those partitions permitted by the policy
 - Required significant restructuring of the untrusted DBMS software to make this work
- Speculation – could HWMAC labels on each byte of data allow an essentially unmodified DBMS to be MLS? (Small modifications might still be needed, but at much lower cost)
 - We have NOT yet studied this question – potentially HUGE payoff if this could work
 - There are covert channel issues in the database metadata that could cause serious problems, but perhaps that can be overcome
 - Significant additional research needed

HWMAC to implement capability architecture and/or HW typing

- Our proposed HWMAC architecture attaches multi-bit tags to each byte of memory
- HWMAC uses the concept of a tag from the early tagged memory approaches, such as the Burroughs B5000 or various tagged capability machines, such as the Chicago Magic Number machine or the IBM System/38
 - Major difference is that the policy enforced from the tags is dynamically loadable into the CPU hardware, rather than simply implemented in a software type manager
 - Our tags are per-byte, while the traditional capability machines tagged per word
- HWMAC tags can be used to distinguish data from capabilities, as in traditional tagged capability machines
 - Note that most PowerPC processors already include a one-bit tag per 64-bit word to provide tagged capability hardware for IBM's System i servers (formerly known as AS/400)
- HWMAC tags could also designate data types, as in the B5000 processor which had only one kind of arithmetic instructions
 - The B5000 tag bits determined whether the operands were fixed point, floating point, etc.

SQL Injection Attacks

- Problem – User Supplied Input:
 - SELECT * FROM users WHERE name='David'
 - SELECT * FROM users WHERE name='a';DROP TABLE users;'
- Most widely used exploits in the wild today are either SQL or shell injection attacks

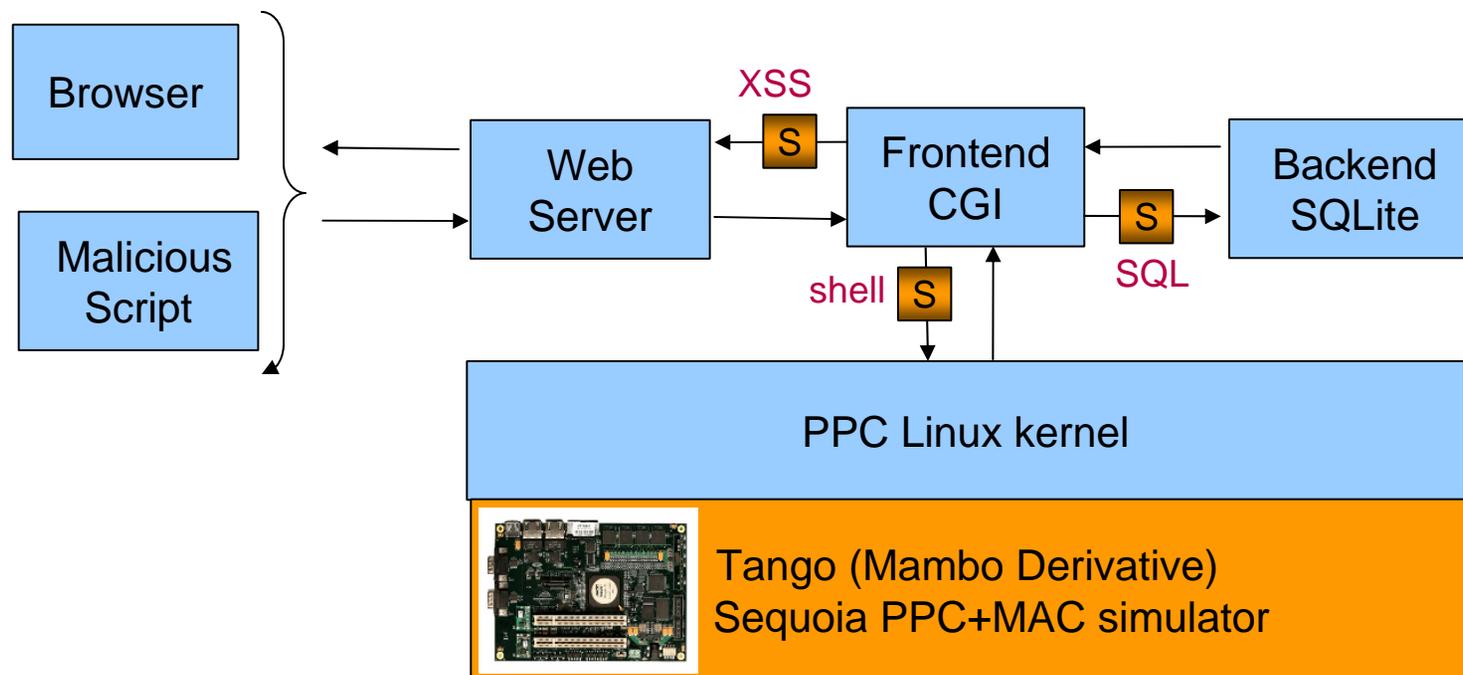
Injection Attacks (SQL, Shell, XSS)

- Potential Solutions:
 - **Fix API** (use parameterized calls- `execve()`, parameterized SQL)
 - Lots of legacy code to fix
 - **Software “escaping”**
 - `SELECT * FROM users WHERE name='a\';DROP TABLE users\;'`
 - Lots of code to rewrite, easy to miss instances, bypassable
 - **Hardware MAC labelling** – a single mechanism
 - Can cover all layers of software, and interfaces
 - Can automatically trap/remediate at runtime
 - Does not trust software
 - Do not need to change existing code
 - Theoretical Basis for byte label sanitization

Live Demo Categories (28 Total Demonstrations)

- Simulated CPU
 - 440 unmodified
 - 440 with MAC
- Main pages
 - SQL injection
 - Shell Injection
- Options
 - XSS
 - Heap pointer overflow
 - Bypass sanitizer
- We will only show Demos related to SQL injection
- Not enough time for live demo during this presentation
 - Can run it after sessions or during breaks

Injection Sanitization – Demo Architecture



Unmodified

New/Modified

S Sanitizer

Tango is a PPC functional simulator, developed by IBM Research

Sequoia is a third-party embedded PPC board that we simulate in Tango

Truth in advertising: For this demo we implemented the instructions that control the policy as user level instructions to avoid having to rewrite the OS to demonstrate the value of the architecture. These should be privileged instructions

SQL Demo – HTML Form

NGSCA SQL Injection Demo on PowerPC 440EPx 'with MAC'

Lookup by Name

Last Name

First Name

SQL Demo – HTML Form

Your results on 11/16/2009:

David Safford 914-784-6261

(Backend Command was `./sqlite phone.db "select * from phone where fname='David'"`)

SQL Demo – Good Input on MAC

```

=> colors: ( System Input Front San San2 San3 Back Back2 Code XXX )
=> sql_untrust: (lname=&fname=David)
=> sql_frontend: (./cgi_sanitize "/sqlite phone.db "select * from phone where fname=David")
=> cgi_sanitize: (./backend "/sqlite phone.db "select * from phone where fname=David")
=> backend: (./sqlite phone.db "select * from phone where fname=David")

```

Untrusted input



SQL Demo – Malicious Input Script

```
#!/bin/sh
STRING="x' or 'x' = 'x"
NET=192.168.1.30
for opt in $*; do
if [ $opt == "-cgi" ]; then STRING=$STRING\&cgi=f;
elif [ $opt == "cgi" ]; then STRING=$STRING\&cgi=t;
elif [ $opt == "-xss" ]; then STRING=$STRING\&xss=f;
elif [ $opt == "xss" ]; then STRING=$STRING\&xss=t;
elif [ $opt == "pass" ]; then STRING=$STRING\&pol=1;
elif [ $opt == "ignore" ]; then STRING=$STRING\&pol=2;
elif [ $opt == "40" ]; then NET=192.168.1.40;
fi
done
(perl -e 'print "POST /cgi-bin/sql_untrust HTTP/1.0\r\nContent-Type: "';
perl -e 'print "application/x-www-form-urlencoded\r\nContent-Length: "';
perl -e 'print "69\r\n\r\nlname="" ; echo $STRING) | nc $NET 80 ;
```

SQL Demo – Malicious Input on MAC

```

=> colors: ( System Input Front San San2 San3 Back Back2 Code XXX )
=> sql_untrust: (lname=x' or 'x' = 'x)
=> sql_frontend: (./cgi_sanitize "/sqlite phone.db \"select * from phone where lname= x' or 'x' = 'x'\"")
=> cgi_sanitize: (./backend "/sqlite phone.db \"select * from phone where lname='x' or 'x' = 'x'\"")
=> backend: (./sqlite phone.db "select * from phone where lname='x' or 'x' = 'x'")
  
```

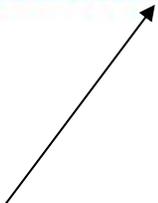
Good quote Bad quote
 Good quote Not escaped Bad quote escaped

SQL Demo – Malicious Input with bypass on MAC

```

=> colors: ( System Input Front San San2 San3 Back Back2 Code XXX )
=> sql_untrust: (lname=x' or 'x' = 'x&pol=1)
=> sql_frontend: (./cgi_sanitize "/sqlite phone.db "select * from phone where lname='x' or 'x' = 'x\'\" 1)
=> cgi_sanitize: (./backend "/sqlite phone.db "select * from phone where lname='x\' or \'x\' = \'x\'\" 1)
=> backend: ( )

```


 Trap on backend reading first “frontend” character.
 This demonstrates byte level integrity MAC.

Conclusions

- We have demonstrated that significant security benefits can be gained from a major new approach to hardware security features
- HWMAC proposes significant changes to the CPU architectures
 - Significant additional logic to implement policy engines
 - Potentially doubling memory consumption to store per-byte tags in memory and on disk
 - New software loadable policy tables
- Mitigating factors
 - Memory is cheap!
 - Processor logic is also cheap, given that software still does not exploit massive parallelism well, except in a few special cases
- This is a snapshot of high risk, potentially high payoff research in progress
- HWMAC is intended to be a prototype used to explore this design space.
 - HWMAC seems to be a very powerful mechanism
- Open research questions:
 - Do the security benefits gained outweigh the costs?
 - Management of policy tables may be very hard
 - Is this level of flexibility, one policy table per instruction per task required, can we simplify?
 - What else can be done with it that we haven't thought of yet?

Backup Slides

Cited References

- Message segments
 - Whitmore, J., A. Bensoussan, P. Green, D. Hunt, A. Kobziar, and J. Stern, *Design for Multics Security Enhancements*, ESD-TR-74-176, December 1973, Honeywell Information Systems, Inc., HQ Electronic Systems Division: Hanscom AFB, MA. URL: <http://csrc.nist.gov/publications/history/whit74.pdf>
- Hinke-Schaefer
 - Hinke, T.H. and M. Schaefer, *Secure Data Management System*, RADC-TR-75-266 [NTIS AD A019201], November 1975, Rome Air Development Center: Griffiss AFB, NY.
- Seaview
 - Lunt, T.F. and P.K. Boucher. *The SeaView Prototype: Project Summary*. in **17th National Computer Security Conference**. 11-14 October 1994, Baltimore, MD: Vol. 1. National Institute of Standard and Technology / National Computer Security Center. p. 88-102.
- Burroughs B5000
 - Organick, E.I., *Computer System Organization - The B5700/B6700 Series*. 1973, New York: Academic Press. URL: http://bitsavers.org/pdf/burroughs/B5000_5500_5700/Organick_B5700_B6700_1973.pdf
- IBM iSeries
 - Soltis, F.G., *Fortress Rochester : The Inside Story of the IBM iSeries*. 2001, Lewisville, TX: 29th Street Press