Lecture 22, May 20, 2026

ECS 235B, Foundations of Computer and Information Security

Spring Quarter 2026

Analyzing Covert Channels

- Policy and operational issues determine how dangerous it is
  - What follows assumes a policy saying all covert channels are a problem
- *Amount* of information that can be transmitted affects how serious a problem a covert channel is
  - 1 bit per hour: probably harmless in most circumstances
  - 1,000,000 bits per second: probably dangerous in most circumstances
  - Begin here . . .

Measuring Capacity

- Intuitively, difference between unmodulated, modulated channel
  - Suppose normal uncertainty in channel is 8 bits
  - Attacker modulates channel to send information, reducing uncertainty to 5 bits
  - Covert channel capacity is 3 bits
    - Modulation in effect fixes those bits

Formally

- Inputs:
  - *A* input from Alice (sender)
  - *V* input from everyone else
  - *X* output of channel
- Capacity measures uncertainty in *X* given *A*
- In other terms: maximize

$$I(A; X) = H(X) - H(X \mid A)$$

  with respect to *A*

Noninterference and Covert Channels

- If *A*, *V* are independent and *A* noninterfering with *X*, then *I*(*A*; *X*) = 0
- Why? Intuition is that *A* and *X* are independent
  - If so, then only *V* affects *X* (noninterference)
  - So information from *A* cannot affect *X* unless *A* influences *V*
  - But *A* and *V* are independent, so information from *A* does not affect *X*
- But noninterference is not necessary

Example: Noninterference Not Necessary

- System has 1 bit of state; 3 inputs I_A, I_B, I_C; one output O_X
- Each input flips state, and state's value is then output
  - System initially in state 0
- w sequence of inputs corresponding to output $x(w) = length(w) \bmod 2$
  - I_A not noninterfering as deleting its inputs may change output
- Define terms
  - W random variable corresponding to length of input sequences
  - A random variable corresponding to length of input sequences contributed by I_A; V random variable corresponding to other contributions; A, V independent
  - X random variable corresponding to output state

Two Cases

- $V = 0$; then as $W = (A + V) \bmod 2$, $W = A$, and so A, W not independent; neither are A, X. So if $V = 0$, $I(A, X) \neq 0$
- I_B, I_C produce inputs such that p(V=0) = p(V=1) = 0.5; then

$$p(X=x) = p(V=x, A=0) + p(V = 1 - x, A = 1)$$

Because A, V independent, this becomes

$$p(X=x) = p(V=x, A=0) + p(V = 1 - x)p(A = 1)$$

and so $p(X=x) = 0.5$. Also,

$$p(X=x \mid A=a) = p(X = (a + x) \bmod 2) = 0.5$$

establishing A, X independent; so $I(A, X) = 0$

Meaning

- Note A, X noninterfering, and I(A; X) = 0
- So covert channel capacity is 0 if either of the following hold:
  - Input is noninterfering with output; or
  - Input comes from independent sources, all possible values from at least one source are equally probable

Example (More Formally)

- If A, V independent, take $p=p(A=0)$, $q=p(V=0)$:
  - $p(A=0,V=0) = pq$
  - $p(A=1,V=0) = (1-p)q$
  - $p(A=0,V=1) = p(1-q)$
  - $p(A=1,V=1) = (1-p)(1-q)$
- So
  - $p(X=0) = p(A=0,V=0)+p(A=1,V=1) = pq + (1-p)(1-q)$
  - $p(X=1) = p(A=0,V=1)+p(A=1,V=0) = (1-p)q + p(1-q)$

Example (*con't*)

- Also:
  - $p(X=0|A=0) = q$
  - $p(X=0|A=1) = 1-q$
  - $p(X=1|A=0) = 1-q$
  - $p(X=1|A=1) = q$
- So you can compute:
  - $H(X) = -[(1-p)q + p(1-q)] \lg [(1-p)q + p(1-q)]$
  - $H(X|A) = -q \lg q - (1-q) \lg (1-q)$
  - $I(A;X) = H(X) - H(X|A)$

Example (*con't*)

- So $I(A; X) = -[pq + (1-p)(1-q)] \lg [pq + (1-p)(1-q)] -$
  $[(1-p)q + p(1-q)] \lg [(1-p)q + p(1-q)] +$
  $q \lg q + (1-q) \lg (1-q)$
- Maximum when p = 0.5; then
  $$I(A;X) = 1 + q \lg q + (1-q) \lg (1-q) = 1 - H(V)$$
- So, if $q = 0$ (meaning V is constant) then $I(A;X) = 1$
- Also, if $q = p = 0.5$, $I(A;X) = 0$

Analyzing Capacity

- Assume a noisy channel
- Examine covert channel in MLS database that uses replication to ensure availability
  - 2-phase commit protocol ensures atomicity
  - *Coordinator* process manages global execution
  - *Participant* processes do everything else

How It Works

- Coordinator sends message to each participant asking whether to abort or commit transaction
  - If any says “abort”, coordinator stops
- Coordinator gathers replies
  - If all say “commit”, sends commit messages back to participants
  - If any says “abort”, sends abort messages back to participants
  - Each participant that sent commit waits for reply; on receipt, acts accordingly

Exceptions

- Protocol times out, causing party to act as if transaction aborted, when:
  - Coordinator doesn’t receive reply from participant
  - Participant who sends a commit doesn’t receive reply from coordinator

Covert Channel Here

- Two types of components
  - One at *Low* security level, other at *High*
- Low component begins 2-phase commit
  - Both *High*, *Low* components must cooperate in the 2-phase commit protocol
- *High* sends information to *Low* by selectively aborting transactions
  - Can send abort messages
  - Can just not do anything

Note

- If transaction *always* succeeded except when *High* component sending information, channel not noisy
  - Capacity would be 1 bit per trial
  - But channel noisy as transactions may abort for reasons *other* than the sending of information

Analysis

- *X* random variable: what *High* user wants to send
  - Assume abort is 1, commit is 0
  - $p = p(X=0)$ probability *High* sends 0
- *A* random variable: what *Low* receives
  - For noiseless channel $X = A$
- n+2 users
  - Sender, receiver, n others that act independently of one another
  - q probability of transaction aborting at any of these n users

Basic Probabilities

- Probabilities of receiving given sending
  - $p(A=0|X=0) = (1-q)^n$
  - $p(A=1|X=0) = 1-(1-q)^n$
  - $p(A=0|X=1) = 0$
  - $p(A=1|X=1) = 1$
- So probabilities of receiving values:
  - $p(A=0) = p(1-q)^n$
  - $p(A=1) = 1-p(1-q)^n$

More Probabilities

- Given sending, what is receiving?
  - $p(X=0|A=0) = 1$
  - $p(X=1|A=0) = 0$
  - $p(X=0|A=1) = p[1–(1–q)^n] / [1–p(1–q)^n]$
  - $p(X=1|A=1) = (1–p) / [1–p(1–q)^n]$

Entropies

You can compute these:

- $H(X) = -p \lg p - (1-p) \lg (1-p)$
- $H(X|A) = -p[1-(1-q)^n] \lg p - p[1-(1-q)^n] \lg [1-(1-q)^n] + [1-p(1-q)^n] \lg [1-p(1-q)^n] - (1-p) \lg (1-p)$
- $I(A;X) = -p(1-q)^n \lg p + p[1-(1-q)^n] \lg [1-(1-q)^n] - [1-p(1-q)^n] \lg [1-p(1-q)^n]$

Capacity

- Maximize this with respect to p (probability that *High* sends 0)
  - Notation: $m = (1-q)^n$, $M = (1-m)^{(1-m)}$
  - Maximum when $p = M^{(1/m)} / (M^{(1/m)}m+1)$
- Capacity is:

$$I(A;X) = \frac{Mm \lg p + M(1-m) \lg (1-m) + \lg (Mm+1)}{(Mm+1)}$$

Mitigation of Covert Channels

- Problem: these work by varying use of shared resources
- One solution
  - Require processes to say what resources they need before running
  - Provide access to them in a way that no other process can access them
- Cumbersome
  - Includes running (CPU covert channel)
  - Resources stay allocated for lifetime of process

Alternate Approach

- Obscure amount of resources being used
  - Receiver cannot distinguish between what the sender is using and what is added
- How? Two ways:
  - Devote uniform resources to each process
  - Inject randomness into allocation, use of resources

Uniformity

- Variation of isolation
  - Process can’t tell if second process using resource
- Example: KVM/370 covert channel via CPU usage
  - Give each VM a time slice of fixed duration
  - Do not allow VM to surrender its CPU time
    - Can no longer send 0 or 1 by modulating CPU usage

Randomness

- Make noise dominate channel
  - Does not close it, but makes it useless
- Example: MLS database
  - Probability of transaction being aborted by user other than sender, receiver approaches 1
    - $q \rightarrow 1$
  - $I(A; X) \rightarrow 0$
  - How to do this: resolve conflicts by aborting increases q, or have participants abort transactions randomly

Problem: Loss of Efficiency

- Fixed allocation, constraining use
  - Wastes resources
- Increasing probability of aborts
  - Some transactions that will normally commit now fail, requiring more retries
- Policy: is the inefficiency preferable to the covert channel?

Example

- Goal: limit covert timing channels on VAX/VMM
- “Fuzzy time” reduces accuracy of system clocks by generating random clock ticks
  - Random interrupts take any desired distribution
  - System clock updates only after each timer interrupt
  - Kernel rounds time to nearest 0.1 sec before giving it to VM
    - Means it cannot be more accurate than timing of interrupts

Example

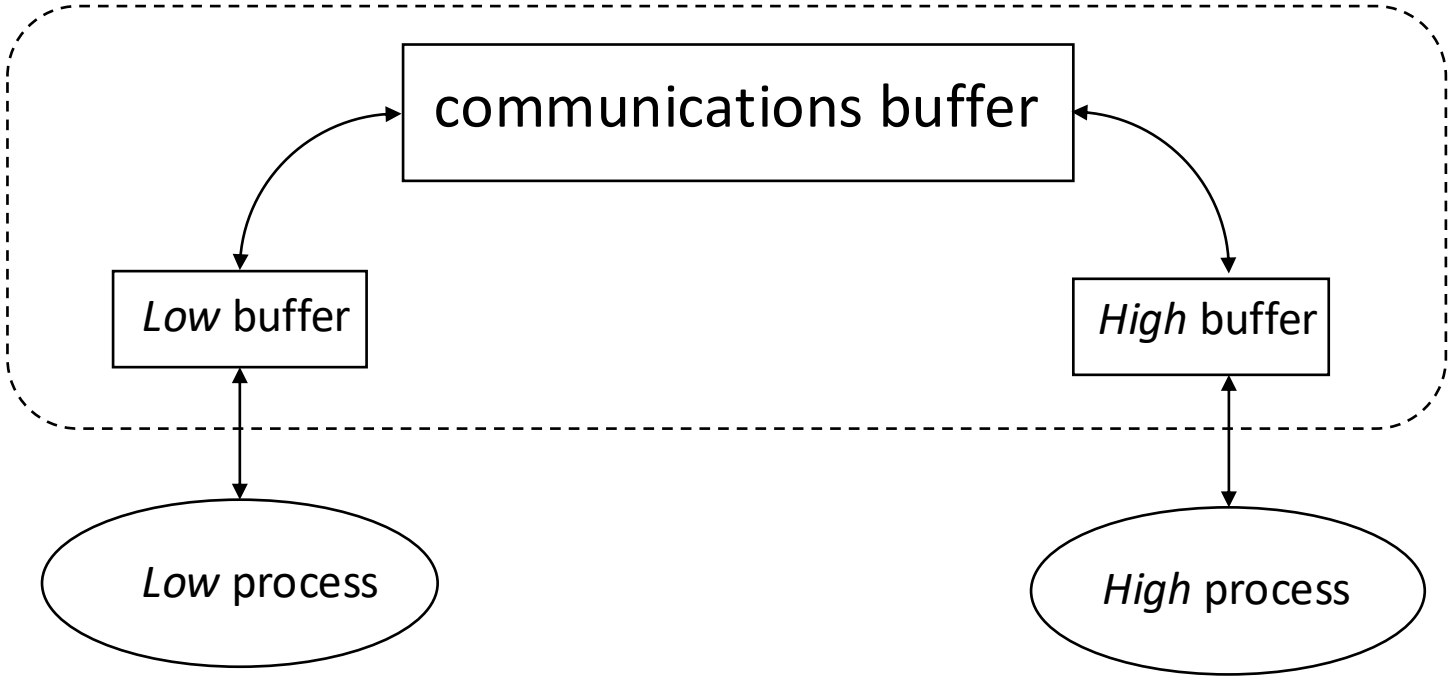
- I/O operations have random delays
- Kernel distinguishes 2 kinds of time:
  - *Event time* (when I/O event occurs)
  - *Notification time* (when VM told I/O event occurred)
    - Random delay between these prevents VM from figuring out when event actually occurred)
    - Delay can be randomly distributed as desired (in security kernel, it's 1–19ms)
  - Added enough noise to make covert timing channels hard to exploit

Improvement

- Modify scheduler to run processes in increasing order of security level
  - Now we're worried about "reads up", so …
- Countermeasures needed only when transition from *dominating* VM to *dominated* VM
  - Add random intervals between quanta for these transitions

The Pump

- Tool for controlling communications path between *High* and *Low*



Details

- Communications buffer of length *n*
  - Means it can hold up to *n* messages
- Messages numbered
- Pump ACKs each message as it is moved from *High* (*Low*) buffer to communications buffer
- If pump crashes, communications buffer preserves messages
  - Processes using pump can recover from crash

Covert Channel

- Low fills communications buffer
  - Send messages to pump until no ACK
  - If *High* wants to send 1, it accepts 1 message from pump; if *High* wants to send 0, it does not
  - If *Low* gets ACK, message moved from *Low* buffer to communications buffer $\Rightarrow$ *High* sent 1
  - If *Low* doesn't get ACK, no message moved $\Rightarrow$ *High* sent 0
- Meaning: if *High* can control rate at which pump passes messages to it, a covert timing channel

Performance vs. Capacity

- Assume *Low* process, pump can process messages more quickly than *High* process
- L_i random variable: time from *Low* sending message to pump to *Low* receiving ACK
- H_i random variable: average time for *High* to ACK each of last *n* messages

Case1: $E(L_i) > H_i$

- *High* can process messages more quickly than *Low* can get ACKs
- Contradicts above assumption
  - Pump must be delaying ACKs
  - *Low* waits for ACK whether or not communications buffer is full
- Covert channel closed
- Not optimal
  - Process may wait to send message even when there is room

Case 2: $E(L_i) < H_i$

- *Low* sending messages faster than *High* can remove them
- Covert channel open
- Optimal performance

Case 3: $E(L_i) = H_i$

- Pump, processes handle messages at same rate
- Covert channel open
  - Bandwidth decreased from optimal case (can't send messages over covert channel as fast)
- Performance not optimal

Adding Noise

- Shown: adding noise to approximate case 3
  - Covert channel capacity reduced to 1/*nr* where *r* time from *Low* sending message to pump to *Low* receiving ACK when communications buffer not full
  - Conclusion: use of pump substantially reduces capacity of covert channel between *High*, *Low* processes when compared to direct connection

Formal Methods Outline

- Formal verification techniques
- Design verification languages
- Bell-LaPadula and SPECIAL
- Current verification systems
- Functional programming languages
- Formally verified products

Formal Verification Techniques

- Formal specification languages for specifying requirements and systems
  - Well-defined semantics, syntax
  - Based on mathematical logic systems
- Mathematically-based automated formal methods for proving properties of specifications and programs
  - Inductive verification techniques
  - Model checking techniques

Inductive Verification vs. Model Checking

Classification criteria:

- *Proof-based vs. model-based techniques*:
  - *premises* embody system description
  - *conclusion* represents properties to be proved
  - Proof-based: derive intermediate formulae that go from premises to conclusion
  - Model-based: establish that premises, conclusion have same truth table values
- *Degree of automation*: fully manual to fully automatic, with everything in between

Inductive Verification vs. Model Checking

Classification criteria:

- *Full vs. property verification*:
  - System specification may describe entire system or part of system
  - Property specification may be single property or many properties
- *Predevelopment vs. postdevelopment*: may be design aid or for verification after system design is complete
- *Intended domain of application*: hardware or software, sequential or concurrent, non-terminating (like an operating system) or terminating, and so forth

Example: HDM

- Developed at SRI
- Began as proof-based formal verification methodology
  - Covers design through implementation
  - Automated, general-purpose methodology
  - Used specification languages, implementation languages
- Provided model checking with its multilevel security tool
  - Input is formal specification in language SPECIAL
  - Theorem prover uses proof-based technique; fully automated property-oriented verification system

Example: HDM

- Tool uses SRI model (interpretation of Bell-LaPadula model)
  - Given a SPECIAL specification
  - Verification condition generator creates formulae that assert specification correctly implements SRI model
  - Boyer-Moore theorem prover processes these formulae
  - Output is list of the formulae that were satisfied and those that were not

Formal Specification

- A specification written in a formal language with restricted syntax, well-defined semantics, based on well-established mathematical concepts
  - Precise semantics avoids ambiguity
  - Languages support exact descriptions of system function behavior
  - Generally eliminate implementation details
- Automated tools support verification of syntax, semantics

Example Language: SPECIAL

- First-order logic-based language
  - Nonprocedural, strongly typed
- Specification in SPECIAL represents module
  - Specifier defines module scope
  - Systems described in terms of modules
- Function representation in modules
  - VFUN: describe variable data
  - OFUN: describe state transitions
  - OVFUN: describe state transitions and changes in VFUN values

Bell-LaPadula Model and SPECIAL

MODULE Bell_LaPadula_Model give-access

TYPES

Subject_ID: **DESIGNATOR**;

Object_ID: **DESIGNATOR**;

Access_Mode: {OBSERVE_ONLY, ALTER_ONLY, OBSERVE_AND_ALTER};

Access: **STRUCT_OF**(Subject_ID subject;
Object_ID object;
Access_Mode mode);

Comments

- Subject_ID, Object_ID types described at lower level of abstraction
  - The DESIGNATOR indicates this
- Access_Mode types have 3 possible values
- Access type is structure with 3 fields of types shown

Bell-LaPadula Model and SPECIAL

FUNCTIONS
VFUN active (Object_ID object) -> **BOOLEAN** active:
HIDDEN;
INITIALLY
    TRUE;

VFUN access_matrix () -> Access accesses:
HIDDEN;
INITIALLY
    FORALL Access a: a **INSET** accesses => active(a.object);

Comments

- VFUN *active*(*object*) defines the state variable *active* for the *object* and sets it to **TRUE** initially
  - So state variable for that object is true if the object exists
- VFUN access_matrix() defines the state variable *access_matrix* to be set of triples (*subject*, *object*, *right*)
  - This is simply the current set of access rights in the system

Bell-LaPadula Model and SPECIAL

OFUN give-access(Subject_ID giver; Access access);

ASSERTIONS

active(access.object) = **TRUE**;

EFFECTS

access_matrix() = access_matrix() **UNION** (access);

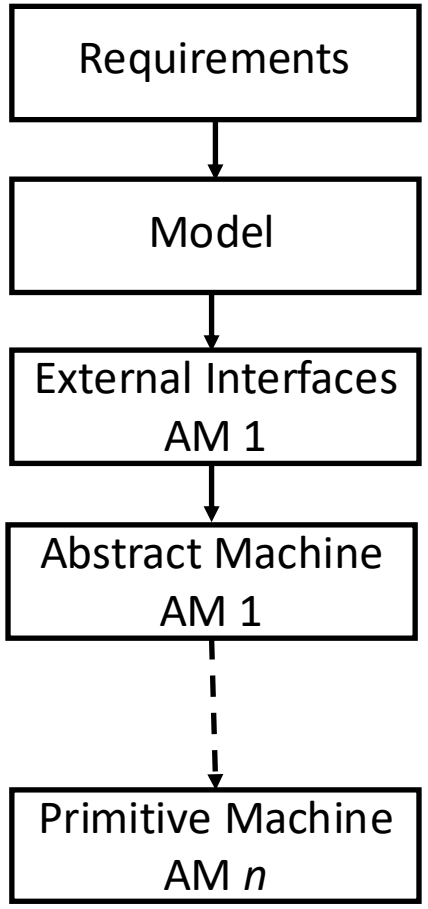
END_MODULE

Comments

- OFUN *access_matrix*() defines state transition when new object added to matrix
- State variable *active* for object must be true
  - See in the **ASSERTIONS** sections
- Value of state variable *access_matrix* after transition is value before transition and additional access rights for the new object

Hierarchical Development Methodology (HDM)

- General-purpose methodology for design, implementation
  - Goal was to automate and formalize development process
- System design specification is hierarchy of a series of abstract machines at increasing level of detail



Analyze, accept requirements

Model proven internally consistent, used as basis for verifying lower AMs

First AM is usually external interface, called Formal Top Level Specification

Each AM mapped to next lower AM, which represents lower levels of system specification

Some combination of hardware and software that runs verified system

Specifications

- *Hierarchical* specification identifies abstract machines (AMs) making up hierarchy
- Each AM a set of modules written in SPECIAL
  - Modules could be reused in more than one AM
- *Mapping specifications* define functions of one AM in terms of next higher AM
- Hierarchy consistency checker: ensured consistency among hierarchy specs, associated module specs for AMs, mapping specs between AMs

Design Hierarchy

- Look at each pair of consecutive AMs, mappings between them
- For each function in higher AM, write programs to show how it was implemented in terms of lower-level AM
  - Written in high-order language
  - Translator mapped program into common internal form that HDM tools used
  - Specs mapped into intermediate language; this and common internal form generated verification conditions
    - Sent to Boyer-Moore theorem prover
  - If lower-level AM correct, then higher-level AM verified to work correctly

Verification in HDM

- Approach: prove the FTLS correctly implemented predefined properties within a model
- Used to verify design of a multi-level security (MLS) tool implementing a version of Bell-LaPadula model (called *SRI model*)

SRI Model

- Some SRI model entities had no corresponding Bell-LaPadula features
  - Visible function references and results (VFUN, OVFUN)
  - Defined subjects implicitly (function callers)
  - *-property addresses downward flow of information
- Bell-LaPadula model had features SRI model did not
  - Discretionary access control, current access triples
  - Defined subjects explicitly
  - *-property addressed allowable downward access

Properties of SRI Model in MLS Tool

- Information returned by specific function invocation to subject can depend only on information with security levels no greater than subject
- Information flowing into state variable (ie, VFUN) can depend only on other state variables with security levels no greater than that of first state variable
- If value of state variable modified, only function invocation with security level no greater than level of state variable can do the modification

MLS Tool

- Processed SPECIAL specification describing external interfaces to SPECIAL model
  - One AM represented, so no mappings
  - Could be multiple modules in specification; each module had to be verified, and then the set verified using hierarchy consistency tool

MLS Tool

- To verify properties:
  - MLS tool generated formulae claiming correctness of properties
  - Property 1 correctness: formulae generated from exceptions from visible functions and VFUN, OVFUN return values
  - Properties 2, 3 correctness: formulae generated for each new value assignment to state variables
- Formulae (*verification conditions*) submitted to theorem prover
- Theorem prover reported the verification conditions that passes, failed, could not be proven

Boyer-Moore Theorem Prover

- User provides theorems, lemmata, axioms, assertions needed for proof
  - For example, rules of reflexivity, associativity, transitivity among partial ordering relations
  - Provided in a LISP-like notation
  - Maintained list of previously proven theorems, axioms for future proofs
- Used extended propositional calculus
- Heuristics organized to find proof in most efficient manner
  - Used a series of steps on formula in search of proof

Boyer-Moore Steps

- *Simplify*: apply axioms, lemmata, function definitions, and other techniques
- *Reformulate*: replace terms by equivalent terms easier to process
- *Substitute equalities*: replace equal expressions with appropriate substitutions to eliminate equality expressions
- *Generalize*: introduce variables for terms that are no longer used
- *Eliminate* irrelevant terms
- *Use induction* to prove theorems when needed

Boyer-Moore Evaluation

1. Iterated between simplify, reformulate steps until formula proved or disproved, or formula did not change
2. Substitute equalities, and if any changes then go back to step 1
3. Generalize, and if any changes then go back to step 1
4. Eliminate, and if any changes then go back to step 1
5. Apply induction, and if any changes then go back to step 1

If formula reduced to **TRUE** or **FALSE**, done; otherwise formula could not be proven

Enhanced HDM (EHDM)

EHDM addressed difficulties with HDM

1. SPECIAL not defined in terms Boyer-Moore theorem prover could use readily
   - Missing specific constructs that theorem prover needed
   - EHDM used new language, similar to SPECIAL but with the missing constructs, such as concepts of AXIONM, THEOREM, LEMMA
2. HDM theorem prover not interactive
   - EHDM theorem prover based on Boyer-Moore theorem prover, but was interactive

Gypsy Verification Environment

- Gypsy Verification Environment (GVE) focused on implementation proofs
  - Verification system tried to show correspondence between specifications, their implementation
  - Verification system could also prove properties of Gypsy specifications
- Set of tools including a Gypsy language parser, verification condition generator, theorem prover

Gypsy Language

- Combined specification language constructs with programming language (Pascal base)
- Limitations on Pascal base
  - Could not nest routines, but could group them together in named ”scope”
  - No global variables; only constants, types, functions, procedures visible between routines
  - Parameters all constant and passed only by reference
  - No pointers
  - New data structures sets, sequences, mappings, buffers; new operations of addition, deletion, moving component