BitBlaze: Binary Analysis for OS Security

Stephen McCamant
(for Dawn Song)

Computer Science Dept.
UC Berkeley

Outline

- **BitBlaze overview**
- **Example: analyzing an off-the-shelf OS**
- **Applications to OS security**
  - **Symbolic execution for a kernel & device drivers**
  - **Code hardening through binary rewriting**
- **Data-oriented computing enforcement**

The BitBlaze Approach

- **Semantics based, focus on root cause:**

  Automatically extracting security-related properties from binary code (vulnerable programs & malicious code) for effective defense

- **Automatically create high-quality detection & defense mechanisms**

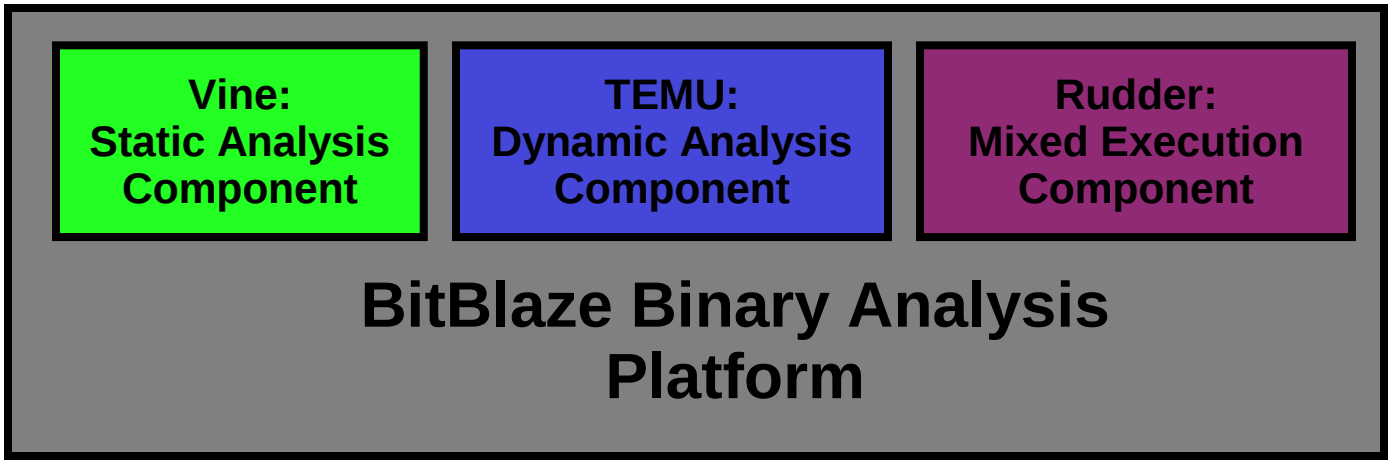
- **Able to handle binary-only setting**
  - **Important for COTS & malicious code scenarios**
  - **Our main target is Windows (where the threat is greatest)**
  - **Binary is truthful**

BitBlaze Binary Analysis Infrastructure: Challenges

- **Complexity**
  - **Intel manuals for x86 instruction set weigh over 11 pounds**
- **Lack higher-level semantics**
  - **Even disassembling is non-trivial**
- **Require whole-system view**
  - **Operations within kernel and interactions btw processes**
- **Malicious code may obfuscate**
  - **Packing, code encryption, dynamic code generation, etc.**

BitBlaze Binary Analysis Infrastructure: Architecture

- **The first infrastructure:**
  - **Novel fusion of static, dynamic analysis techniques, and formal analysis techniques such as symbolic execution & abstract interpretation**
  - **Capable of analyzing whole system (including OS kernel)**
  - **Capable of analyzing packed/encrypted/obfuscated code**



BitBlaze in Action

- Detect and classify malware
  - Spyware, key logger, rootkit, etc.
- Generate vulnerability signatures
- Understand vulnerabilities based on patch
  - Can even automatically generate exploit
- Infer models of security-sensitive functionality
- Infer protocol grammars
- Measure information flows
- Detect OS hooking behavior (next slide)

Example: Kernel Hooking

- A hook is a modification to the OS state made to redirect execution to malware
- Examples: keylogger, network sniffer
- Challenge: how to identify and understand the hooking mechanism, when any function pointer could be a hook point?

- Approach: fine-grained impact analysis based on intrinsic behavior
- No prior knowledge of mechanism required
- Applied to Windows XP SP2
- Replaced manual analysis for 8 malware samples (e.g., Sony BMG DRM rootkit)

Application Directions

- **Symbolic execution for the kernel and its device drivers**

- **Hardening off-the-shelf code with binary rewriting**

Symbolic Execution in the Kernel

- **Mixed concrete and symbolic execution has proved a powerful tool in user-space code security**
  - **Detect behavior deviations, recover models, create signatures, measure information leaks, …**
- **Challenge: can we get the same capabilities for the kernel and device drivers?**

- **Leverage BitBlaze's whole system emulator**
- **Symbolically model system instructions**
- **Key point: concise but sufficiently accurate model of multiple address spaces to enable decision procedure reasoning**

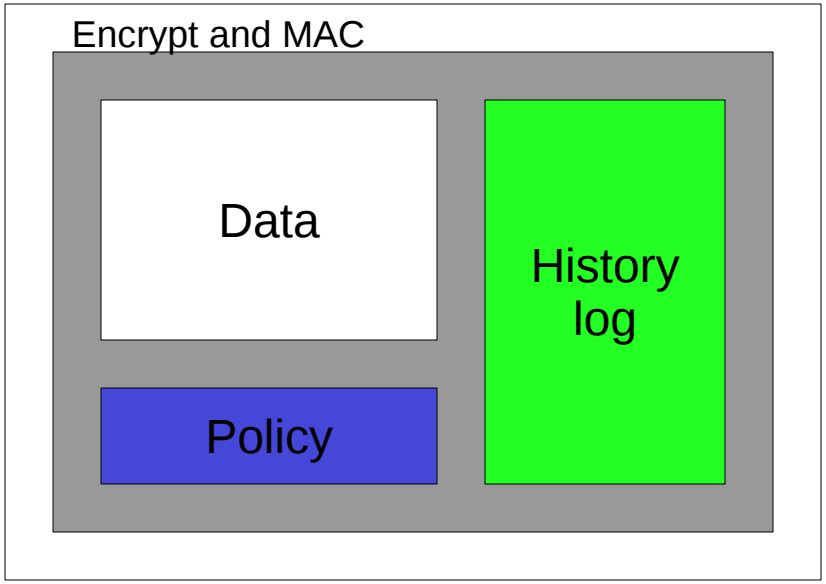
Binary Rewriting Off-the-Shelf Software

- **Enforce more robust protection for commodity software**
- **Challenge: how to handle unstructured control flow, without high level language information?**

- **Combine static and dynamic information sources:**
  - **Extract structural information by comparing related versions**
  - **Record most jump targets during profiling**
  - **Fall back on dynamic rewriting for correctness**

Data-Oriented Computing Enforcement

- **A data-oriented software architecture ties sensitive data to the policy that controls its use**
- **Only trusted code allowed to access**



- **Build on HW and language techniques to enforce**

Isolation

- **Unauthorized code, including compromised OS, must not have access to protected data**
- **Combine mechanisms to get performance, defense in depth**
  - **Binary rewriting**
  - **Language/SVA based type-safety**
  - **Certified code**
  - **Hardware protection**

Descriptive Policies

- For flexibility, want to restrict uses according to code properties, not just code identity
- E.g., work with any video card driver, as long as it does not leak data intended for screen
- Can use either:
  - Code certification: no runtime overhead, precise policies
  - Dynamic enforcement: minimize developer effort, best compatibility
  - Hybrid of the two

Also Required

- Hardware root of trust (e.g., TPM)
  - Thwart complete-virtualization attacks
  - Provide secure key storage
- Encryption/MAC
  - Protect data on untrusted storage media, networks

Summary, Contacts

- **BitBlaze: a unified binary analysis platform for a broad spectrum of security applications**
- **Applications: symbolic execution, binary hardening, data-oriented computing**

- **http://bitblaze.cs.berkeley.edu**

- **dawnsong@cs.berkeley.edu**

- **smcc@cs.berkeley.edu**

Backup slides