TCIPG

Trustworthiness Enhancement Tools for SCADA Software and Platforms

CrossingGuard: Protecting Perimeter-Crossing Buses

Peter C. Johnson and Sergey Bratus

GOALS

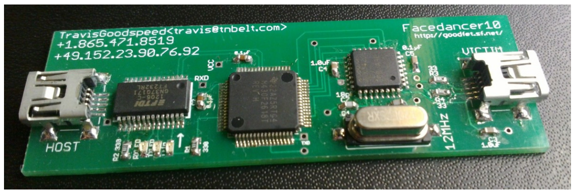
- We aim to protect implementations of bus protocols, on which security of SCADA systems strongly depends;
- To protect **bus** input-handling code on bus boundaries to the same standards to which **network**-facing input-handling code is protected;
- To develop practical secure coding methodologies for bus-facing code;
- To develop a Bus Discipline Kit, for use by bus-facing code such as drivers and/or applications, for safe unmarshalling of bus packets; and
- To integrate with Dartmouth's Hardened Trust Stack for Control Systems.

FUNDAMENTAL QUESTIONS/CHALLENGES

- Semantics of bus protocols have not been scrutinized as closely as those of network protocols. Bus protocol implementations create a large attack surface once an adversary has access to the bus.
- Input-handling code on bus boundaries is a prime target for attackers, yet it lacks mitigations that are now common against network traffic.
- While network traffic is nowadays deeply inspected and firewalled at the perimeter before it hits vulnerable SCADA kernel or application code, inputs coming in on buses enter the kernel **directly**. Any vulnerability in the handling of bus incoming data would **bypass** any layers of protection available for network input.
- Network stacks and bus stacks have similar functionalities: transforming serial data streams into well-typed structures for application layer abstractions. It took a decade for safer programming practices to form in network code; little such development has taken place for bus-facing code. Bus stacks are just as vulnerable as network code was in the 1990s.
- **Firewalls** were a mitigation for network traffic, characterized by *lightweight models of messages and stateful sessions*. Bus sessions have **more state** => more **trust assumptions** between code units => more potential vulnerabilities.

RESEARCH PLAN

- Start with the USB stack (ubiquitous, common target), then generalize methodology to custom stacks and protocols
- Model bus protocol data structures in a domain-specific language (high-level functional language with a developed type system)
- Develop a Domain-Specific Language for bus payloads (BDSL).
- Analyze and instrument existing stack:
  - Design and implement probes for protocol handlers (using a custom DTrace provider or similar)
  - Identify contexts in which payloads of protocol layers can be effectively checked, and implement checking
- Based on the model and instrumentation, develop efficient code for handling and checking bus data.
- Compare generated code and actual stacks for resilience against "fuzzed" payloads.
- Develop examples of data filtering ("firewalling") policies based on the probes and instrumentation. E.g., allow only specific **devices** and **commands** to be processed on the bus, drop or log others.



Facedancer, a custom board for injecting USB packets formulated in Python on the Host to Target, (in collaboration with Travis Goodspeed; hardware and software available from the Dartmouth team)

RESEARCH RESULTS

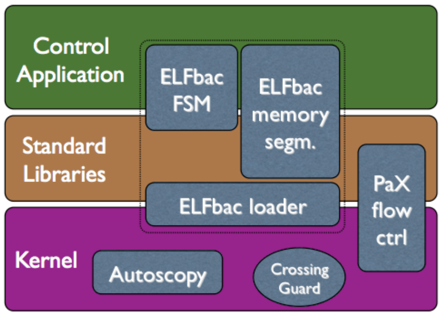
- Instrumentation of the FreeBSD USB stack has been produced and tested.
- A DTrace provider for FreeBSD USB is being produced.
- A prototype model for a subset of the USB protocol has been developed.
- A generative model for USB, USB mass storage, and several subclasses of SCSI commands has been implemented and released.
- Training materials and tools for USB bus payload creation and manipulations have been developed and released.
  - Simple, accessible Python model and code of USB mass storage interface/device.
- Paper "**Perimeter-Crossing Buses**: A New Attack Surface for Embedded Systems" published at the Workshop on Embedded Systems Security (WESS) 2012.

BROADER IMPACT

- On workstations (e.g., an engineer's workstation), buses lack nontrivial security tools and policies. In essence, the alternatives are to disable the USB ports, or risk giving full access to attackers or, worse, insiders. Existing whitelisting can be trivially bypassed with, e.g., a Facedancer device. This problem will persist until a meaningful model of a **firewall architecture** is available. We intend to provide it.
- De-serializing input into complex objects and sessions safely is a hard problem that underlies bugs across all kinds of applications. A successful methodology applied at the bus level will provide a compelling example for other areas.
- Buses and bus-facing software are treated as a special **trusted** case, as if inputs from a bus cannot be maliciously forged in their encoding/frame syntax. We disprove this with the Facedancer device, and show that bus inputs should **not** be treated as trusted.

INTERACTION WITH OTHER PROJECTS

- CrossingGuard will be integrated into the Hardened Stack for Control Systems.



FUTURE EFFORTS

- Develop appropriate framework and components for Linux.
- Extend Bus Discipline Domain-Specific language to other bus protocols.
- Develop driver code and firewall module code generation from the bus data structure/packet specification.
- Combine the above and publish a secure programming methodology for buses.

TRUSTWORTHY CYBER INFRASTRUCTURE FOR THE POWER GRID | TCIPG.ORG
UNIVERSITY OF ILLINOIS | DARTMOUTH COLLEGE | UC DAVIS | WASHINGTON STATE UNIVERSITY
FUNDING SUPPORT PROVIDED BY DOE-OE AND DHS S&T