Segurança de Sistemas e dados (MSI 2021/2022)

Aula 7

Rolando Martins

DCC – FCUP

Slides Adaptados do Prof. Manuel Eduardo Correia

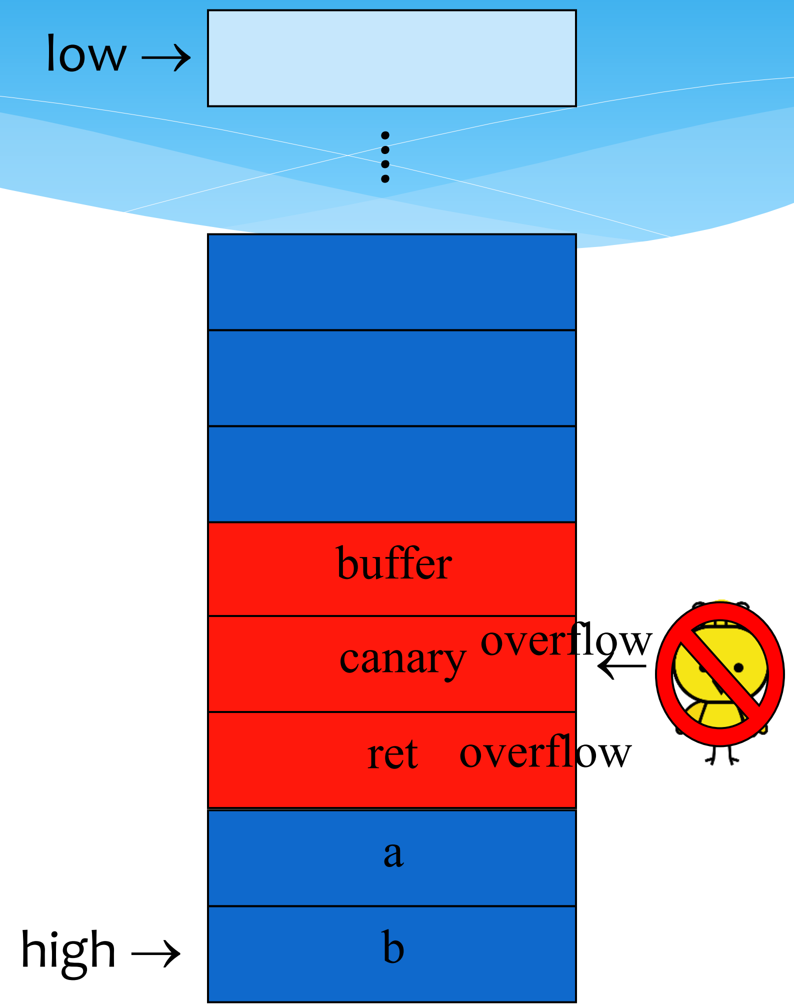
Buffer Overflow (Cont..)

Stack Smashing Prevention

* 1st choice: employ **non-executable stack**
  * "No execute" **NX bit** (if available)
  * Seems like the logical thing to do, but some real code executes on the stack (Java does this)
* 2nd choice: use **safe languages** (Java, Rust, etc)
* 3rd choice: use **safer C functions (only use C if you really need to)**
  * For unsafe functions, there are safer versions
  * For example, strncpy instead of strcpy

Stack Smashing Prevention

* **Canary**
  * Run-time stack check
  * Push canary onto stack
  * Canary value:
    * Constant 0x000aff0d
    * Or value depends on **ret**



Microsoft's Canary

* Microsoft added **buffer security check** feature to C++ with /GS compiler flag
* Uses canary (or "security cookie")
* **Q:** What to do when canary dies?
* **A:** Check for user-supplied handler
* **Handler may be subject to attack**
  * Claimed that attacker can specify handler code
  * If so, "safe" buffer overflows become exploitable when /GS is used!

Buffer Overflow

* Can be prevented
  * Use safe languages/safe functions
  * Educate developers, use tools, etc.
* Buffer overflows will exist for a long time
  * Legacy code
  * Bad software development

Incomplete Mediation

Input Validation

- Consider: `strcpy(buffer, argv[1])`
- A buffer overflow occurs if

  `len(buffer) < len(argv[1])`
- Software must **validate** the input by checking the length of `argv[1]`
- Failure to do so is an example of a more general problem: **incomplete mediation**

Input Validation

* Consider web form data
* Suppose input is validated on client
* For example, the following is valid

```
http://www.things.com/orders/final&custID=112&num=55A&qty=20&price=10&shipping=5&total=205
```

* Suppose input is not checked on server
  * Why bother since input checked on client?
  * Then attacker could send http message

```
http://www.things.com/orders/final&custID=112&num=55A&qty=20&price=10&shipping=5&total=25
```

Incomplete Mediation

- Linux kernel
  - Research has revealed many buffer overflows
  - Many of these are due to incomplete mediation
- Linux kernel is “good” software since
  - Open-source
  - Kernel — written by coding gurus
- Tools exist to help find such problems
  - But incomplete mediation errors can be subtle
  - And tools useful to attackers too!

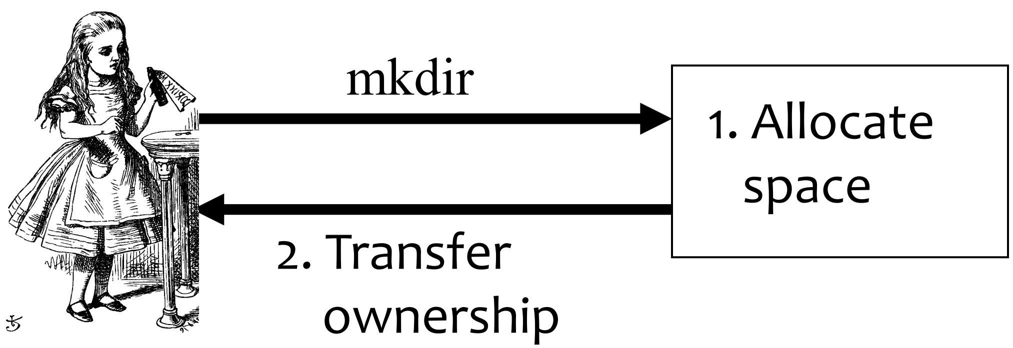
Race Conditions

Race Condition

* Security processes should be **atomic**
  * Occur "all at once"
* Race conditions can arise when security-critical process occurs in stages
* Attacker makes change between stages
  * Often, between stage that gives authorization, but before stage that transfers ownership
* Example: Unix mkdir

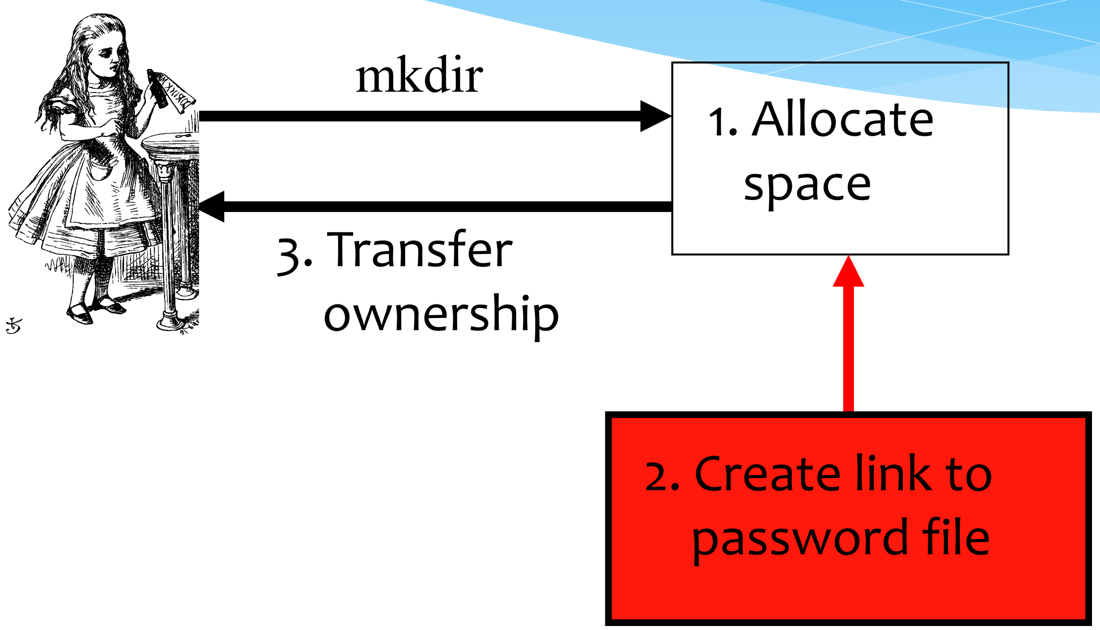
mkdir Race Condition

- mkdir creates new directory
- How mkdir is supposed to work



mkdir Attack

- The mkdir ***race condition***



- Not really a "race"
  - But attacker's timing is critical

Race Conditions

- Race conditions are common
- Race conditions may be more prevalent than buffer overflows
- But race conditions harder to exploit
  - Buffer overflow is "low hanging fruit" today
- To prevent race conditions, make security-critical processes atomic
  - Occur all at once, not in stages
  - Not always easy to accomplish in practice

Race Conditions: Fault Injection

* Hermes and Zermia published on Middleware'13 and NSS'21.
* Fault-injection allow to test corner case scenarios, such as collusion, dDOS and timing attacks.
* Allows to test implementation of complex distributed systems.
* Critical Software has similar approaches for critical systems, more focused on embedded.

Malware

Malicious Software

* Malware is not new…
* Fred Cohen's initial virus work in 1980's
  * Used viruses to break MLS systems
* Types of malware (lots of overlap)
  * **Virus** — passive propagation
  * **Worm** — active propagation
  * Trojan horse — unexpected functionality
  * Trapdoor/backdoor — unauthorized access
  * Rabbit — exhaust system resources

Where do Viruses Live?

* Just about anywhere…
* Boot sector
  * Take control before anything else
* Memory resident
  * Stays in memory
* Applications, macros, data, etc.
* Library routines
* Compilers, debuggers, virus checker, etc.
  * These are particularly nasty!
* And now, firmware!
  * https://www.bleepingcomputer.com/news/security/lenovo-uefi-firmware-driver-bugs-affect-over-100-laptop-models/
* More academic:
  * "Malware in the SGX supply chain: Be careful when signing enclaves!, Vlad Craciun and et al."

Malware Timeline

* Preliminary work by Cohen (early 80's)
* Brain virus (1986)
* Morris worm (1988)
* Code Red (2001)
* SQL Slammer (2004)
* Future of malware?

Brain

- First appeared in 1986
- More annoying than harmful
- A prototype for later viruses
- Not much reaction by users
- What it did
  1. Placed itself in boot sector (and other places)
  2. Screened disk calls to avoid detection
  3. Each disk read, checked boot sector to see if boot sector infected; if not, goto 1
- Brain did nothing malicious

Morris Worm

* First appeared in 1988
* What it tried to do
  * Determine where it could spread
  * Spread its infection
  * Remain undiscovered
* Morris claimed it was a test gone bad
* "Flaw" in worm code — it tried to re-infect infected systems
  * Led to resource exhaustion
  * Adverse effect was like a so-called rabbit

Morris Worm

- How to spread its infection?
- Tried to obtain access to machine by
  - User account password guessing
  - Exploited buffer overflow in fingerd
  - Exploited trapdoor in sendmail
- Flaws in fingerd and sendmail were well-known at the time, but not widely patched

Morris Worm

* Once access had been obtained to machine…
* "Bootstrap loader" sent to victim
  * Consisted of 99 lines of C code
* Victim machine compiled and executed code
* Bootstrap loader then fetched the rest of the worm
* Victim even **authenticated** the sender!

Morris Worm

* How to remain undetected?
* If transmission of the worm was interrupted, all code was deleted
* Code was encrypted when downloaded
* Downloaded code deleted after decrypting and compiling
* When running, the worm regularly changed its name and process identifier (PID)

Result of Morris Worm

- Shocked the Internet community of 1988
  - Internet of 1988 much different than today
- Internet designed to withstand nuclear war
  - Yet it was brought down by a graduate student!
  - At the time, Morris' father worked at NSA…
- Could have been much worse — not malicious
- Users who did not panic recovered quickest
- CERT began, increased security awareness
  - Though limited actions to improve security

Code Red Worm

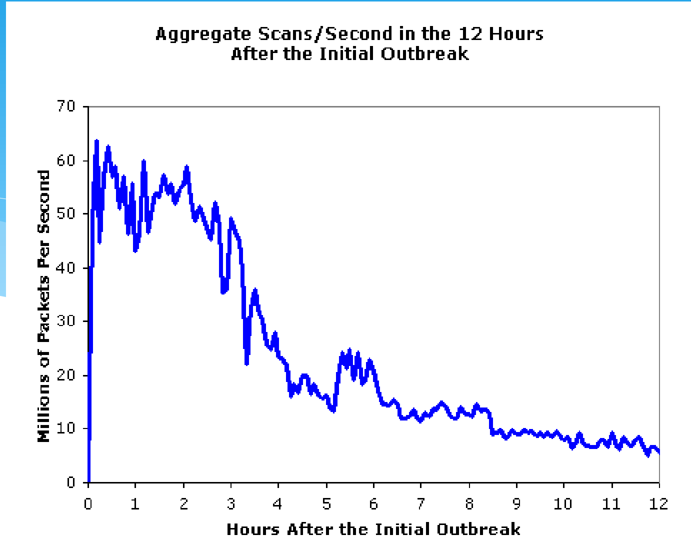
* Appeared in July 2001
* Infected more than **250,000 systems in about 15 hours**
* In total, infected 750,000 out of about 6,000,000 susceptible systems
* **Exploited buffer overflow in Microsoft IIS server software**
* **Then monitored traffic on port 80 for other susceptible servers**

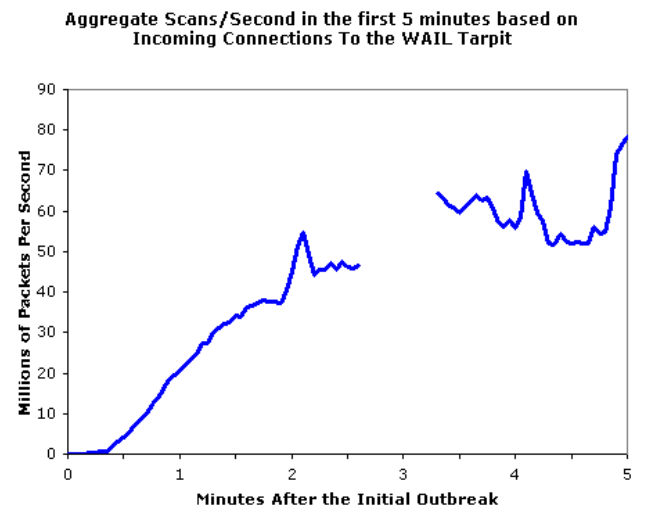
Code Red Worm

* What it did
  * Day 1 to 19 of month: tried to spread infection
  * Day 20 to 27: distributed denial of service attack on `www.whitehouse.gov`
* Later versions (several variants)
  * Included trapdoor for remote access
  * Rebooted to flush worm, leaving only trapdoor
* Has been claimed that Code Red may have been "beta test for information warfare"

SQL Slammer

- Infected **250,000 systems in 10 minutes!**
- Code Red took 15 hours to do what Slammer did in 10 minutes
- At its peak, Slammer infections doubled every 8.5 seconds
- Slammer spread too fast
- “Burned out” available bandwidth



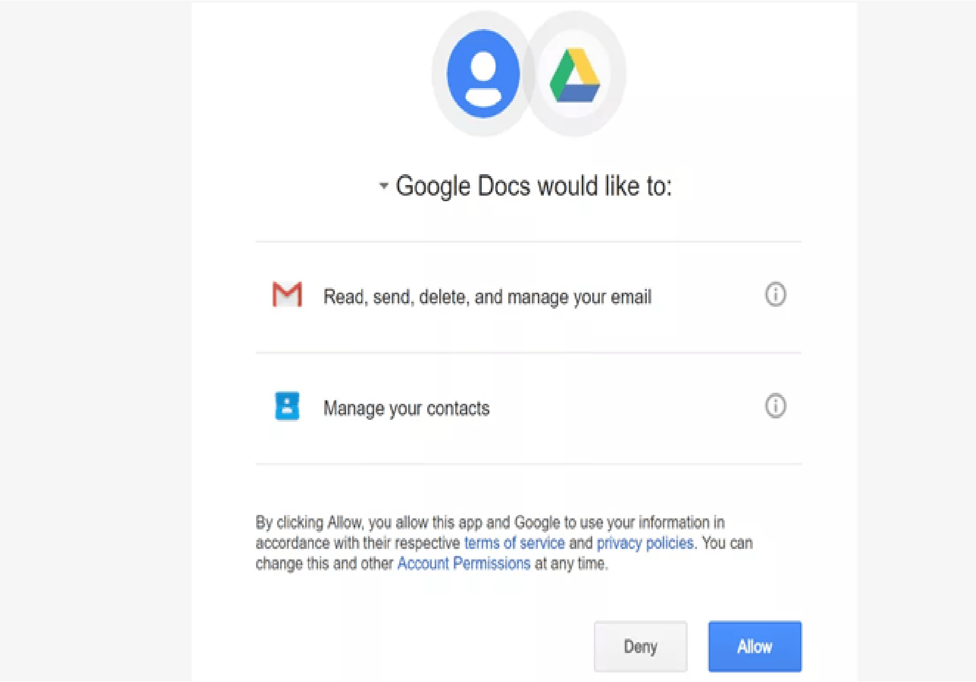


SQL Slammer

* Why was Slammer so successful?
  * Worm fit in **one 376 byte UDP packet**
  * **Firewalls often let small packet thru, assuming it could do no harm by itself**
    * Then firewall monitors the connection
  * Expectation was that much more data would be required for an attack
  * Slammer defied assumptions of "experts"

Trojan

* New insidious Google Docs phishing scheme is rapidly spreading on the web
* May 4, 2017 - http://securityaffairs.co/wordpress/58725/cyber-crime/google-docs-phishing-scheme.html



Malware Detection

* Three common methods
  * Signature detection
  * Change detection
  * Anomaly detection
* We'll briefly discuss each of these
  * And consider advantages and disadvantages of each

Signature Detection

* A **signature** is a string of bits found in software (or could be a hash value)
* Suppose that a virus has signature 0x23956a58bd910345
* We can search for this signature in all files
* If we find the signature are we sure we've found the virus?
  * No, same signature could appear in other files
  * But at random, chance is very small: $1/2^{64}$
  * Software is not random, so probability is higher

Signature Detection

* Advantages
  * Effective on “traditional” malware
  * Minimal burden for users/administrators
* Disadvantages
  * Signature file can be large (10,000’s)…
  * …making scanning slow
  * Signature files must be kept up to date
  * Cannot detect unknown viruses
  * Cannot detect some new types of malware
* By far the most popular detection method

Change Detection

* Viruses must live somewhere on system
* If we detect that a file has changed, it may be infected
* How to detect changes?
  * Hash files and (securely) store hash values
  * Recompute hashes and compare
  * If hash value changes, file **might** be infected

Change Detection

* Advantages
  * Virtually no false negatives
  * Can even detect previously unknown malware
* Disadvantages
  * Many files change — and often
  * Many false alarms (false positives)
  * Heavy burden on users/administrators
  * If suspicious change detected, then what?
  * Might still need signature-based system

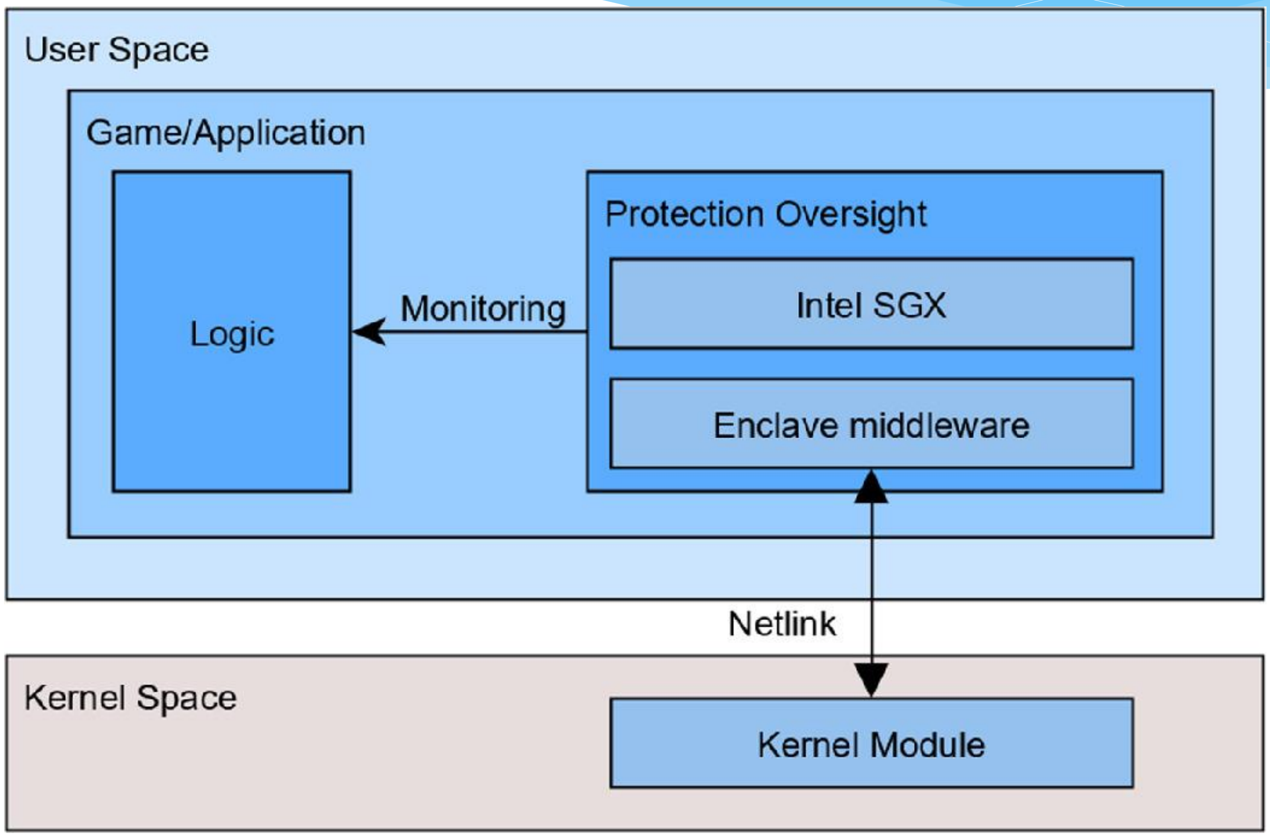
Anomaly Detection

* Monitor system for anything "unusual" or "virus-like" or potentially malicious
* What is unusual?
  * Files change in some unusual way
  * System misbehaves in some way
  * Unusual network activity
  * Unusual file access, etc., etc., etc.
* But must first define "normal"
  * And normal can change!

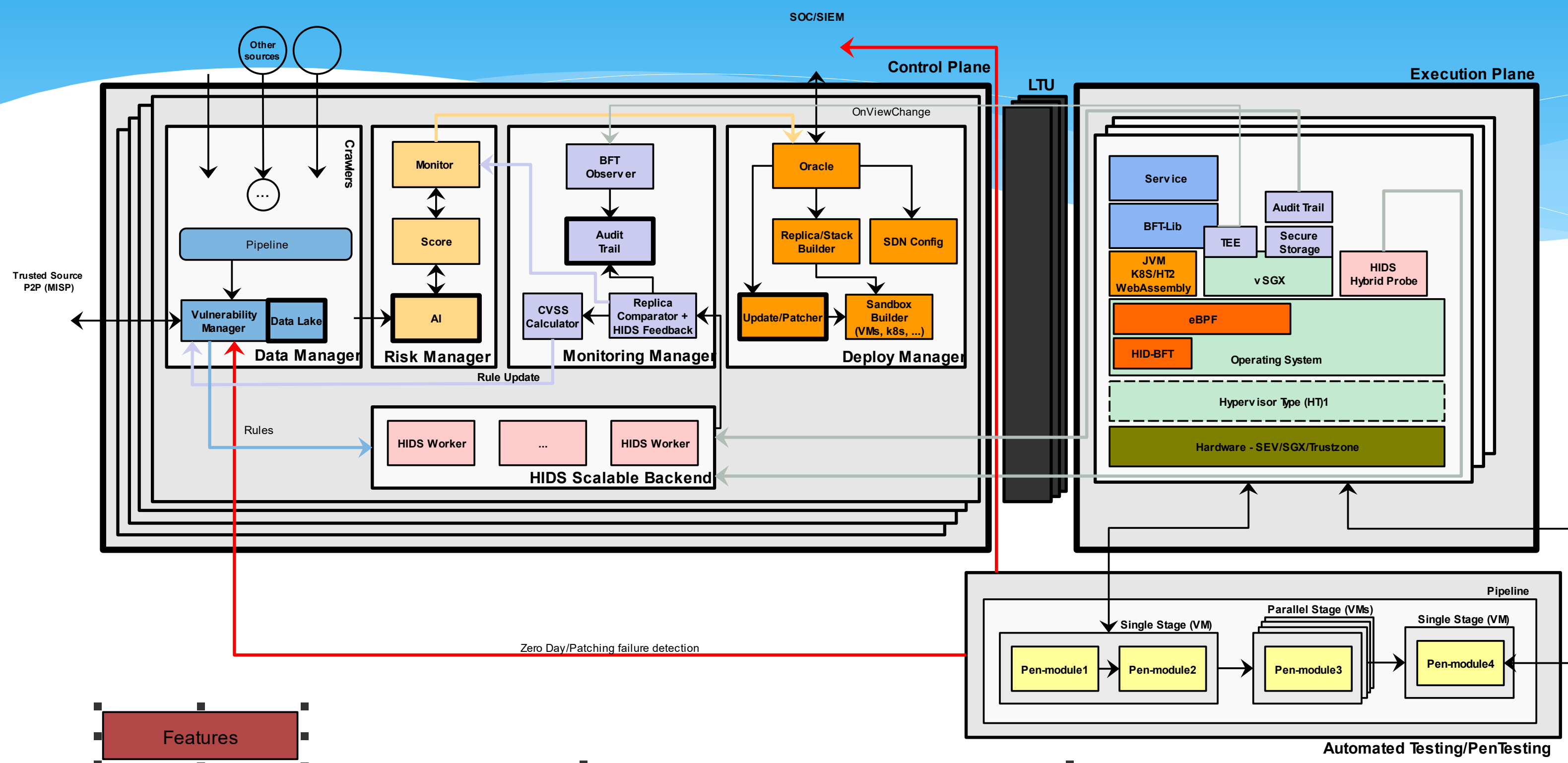
Anomaly Detection

* Advantages
  * Chance of detecting unknown malware
* Disadvantages
  * Unproven in practice
  * Trudy can make abnormal look normal (go slow)
  * Must be combined with another method (such as signature detection)
* Also popular in intrusion detection (IDS)
* A difficult unsolved (unsolvable?) problem
  * As difficult as AI?

Employment of Secure Enclaves in Cheat Detection Hardening, TrustBus’20, André Brandão, João S. Resende & Rolando Martins



Skynet,
Tadeu Freitas' PhD



Features

- Features Zero Trust architecture
- Proactive Self-Healing/Reconfiguration in the presence of Bizantine Failures
- Possible Online Patching - no downtime
- Attaches corporate SOC
- Additional "Exploits Data Lake" as a service
- Artificial intelligence for detecting malicious activies
- Self-Testing

Honeypots

* Are decoy systems
  * filled with fabricated info
  * instrumented with monitors / event loggers
  * divert and hold attacker to collect activity info
  * without exposing production systems
* initially were single systems
* more recently are/emulate entire networks

Honeypot Deployment