Systems and Internet Infrastructure Security

Network and Security Research Center
Department of Computer Science and Engineering
Pennsylvania State University, University Park PA

CSE543 - Introduction to Computer and Network Security
Module: Language Security

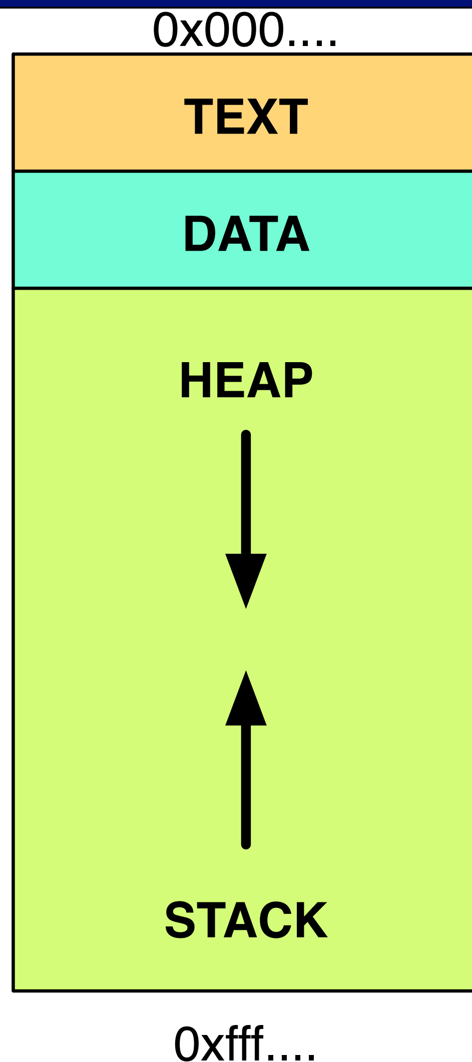
Professor Patrick McDaniel
Fall 2008

Engineering Disaster?

PENNSTATE 1855

- Millions of Bots
  - Compromised applications
- Programming errors
  - Enable code insertion
- What can we do to fix them?
- Just starting to get serious...

Buffer Overflows

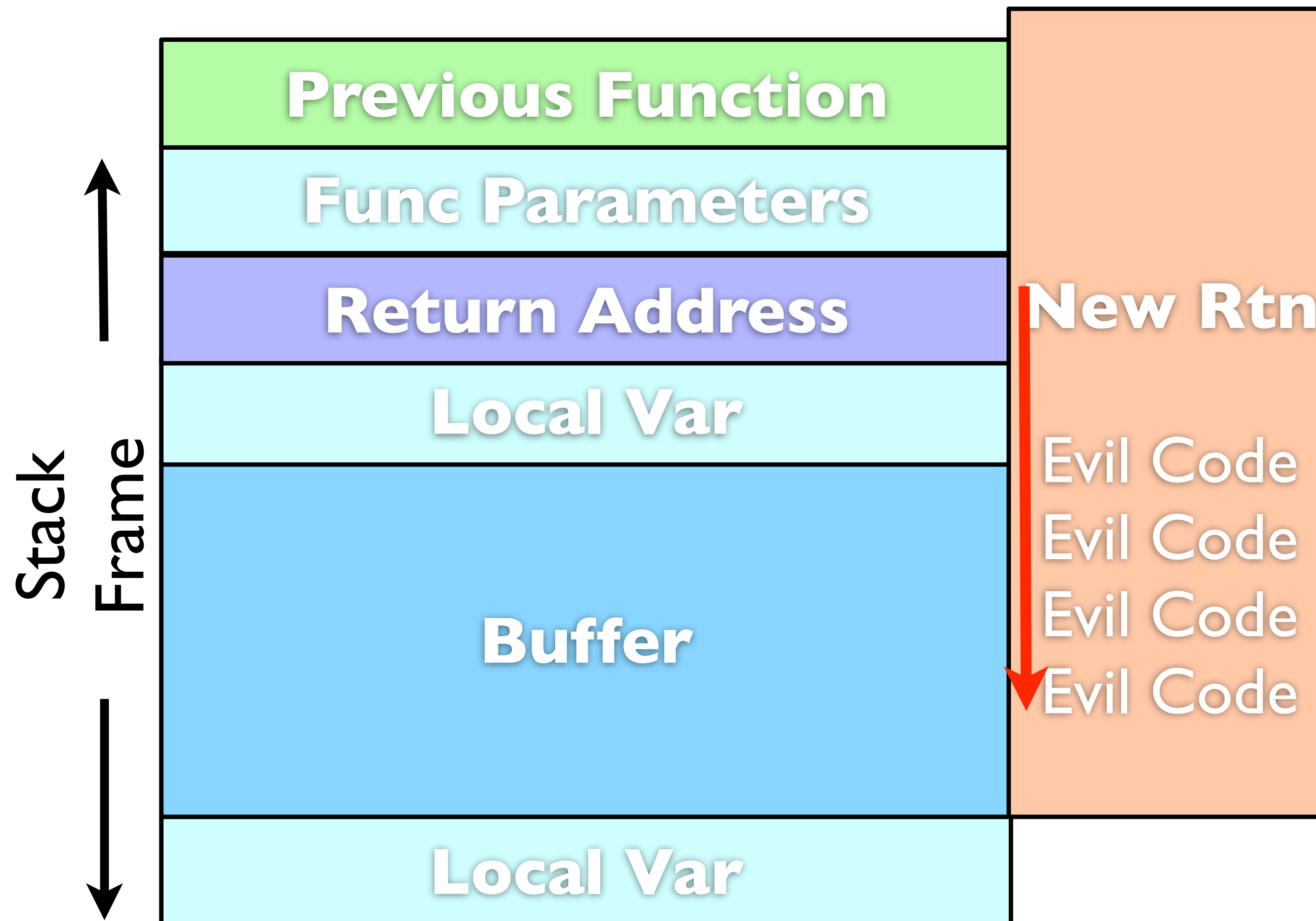


- One means by which the bad guys take over a host
  - install root kits
  - use as SPAM bots
  - use as zombies
  - launch other attacks
- There are many attacks, but this is most prevalent
- It starts with programmer mistake
  - e.g., bad software

Buffer Overflow

PENNSTATE 1855

- How it works



Buffer Overflow Prevention

PENNSTATE 1855

- StackGuard
  - Push a 'canary' on the stack between the local vars and the return pointer
  - Overwrite of canary indicates a buffer overflow
  - Requires changes to the compiler
- Q: Would this solve the problem?
- Thorough summary:
  - www.blackhat.com/presentations/bh-usa-04/bh-us-04-silberman/bh-us-04-silberman-paper.pdf

Other Input Problems

- Function Pointers
  - Overwrite a local function pointer variable
  - Q: What can be done?
- Heap overflow
  - Overflow a buffer on the heap
- Integer Overflow
  - For signed 8-bit integers
    - 127+1 = ??
- Malformed Character Input
  - What does URL "<ipaddr>/scripts/..%c0%af../winnt/system32" decode to?

Java World

- Type Safe Language
  - No buffer/heap/ptr overflows
  - No unsafe casts
  - Still have integer overflows?
- Java Virtual Machine
  - Interpret bytecodes (or compile together)
  - Security Manager (reference monitor for JVM)
- Q: What is the trust model of a Java application?

Ccured

PENNSTATE

- From C to Memory-safe C Translator
  - Find the minimum number of runtime checks to ensure memory safety
- Classify Pointers
  - Safe
  - Wild
    - Need runtime checks for wild pointers
- Runtime Checks
  - Similar to declassifiers in DLM
  - Written by hand, in general

C Analysis

PENNSTATE

- Assume Type Safety in Analysis
  - On what basis?
  - Trust that the programmer does not subvert
- Is this a reasonable assumption?
  - Unsound analysis
    - False negatives are possible
  - Sound analysis
    - If no unsafe behavior relative to analysis can be assumed
- Actually, lots of work in this area
- Used in production code: Microsoft

Source Code Analysis

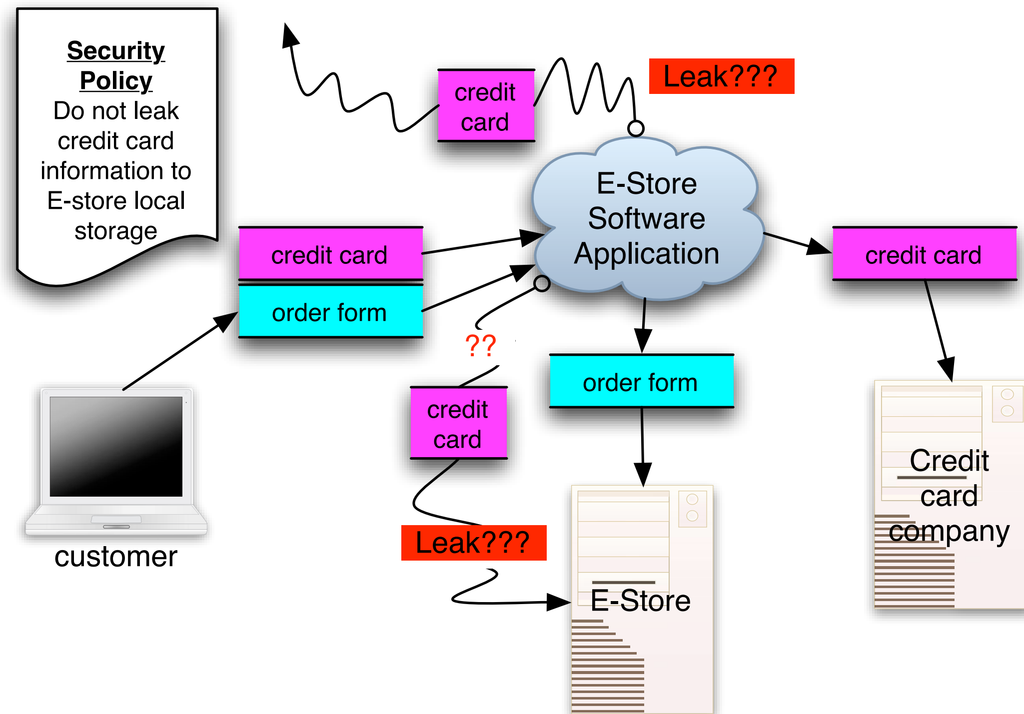
PENNSTATE 1855

- Shallow tools for bug finding
  - Prefix, Prefast -- Microsoft
- Companies that will check your code
  - Coverity -- based on MC
- Deep tools for verifying correctness
  - SLAM -- for device drivers
- Add security to legacy code
  - Generate LSM
  - Generate reference monitor for X Server
- Lots of other topics
  - Privilege separation
  - Domain transition
  - Error reporting

Enforcing security policy

PENNSTATE 1855

- DAC
- MAC
- certificates
- trust management
- SELinux
- anti-virus
- IDS
- firewalls
- encryption
- legal measures

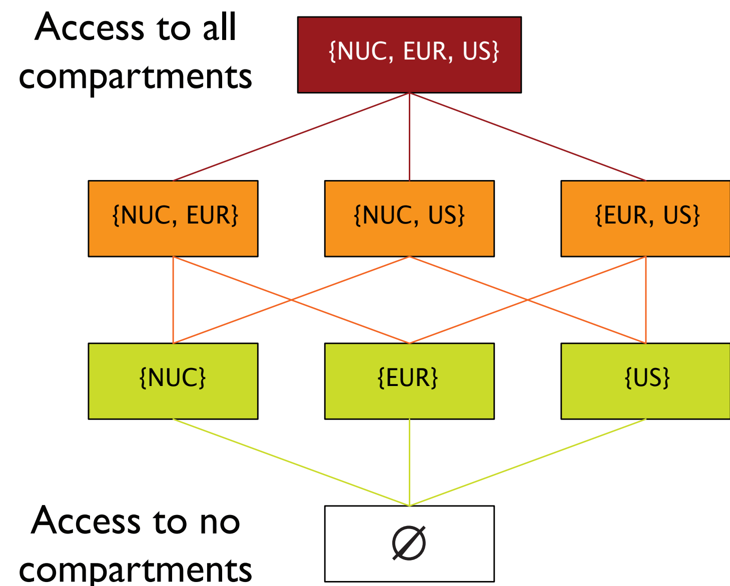


None of these provide end-to-end confidentiality

Information-flow control

PENNSTATE 1855

- What is it?
  - Simple security
  - ★-property



- Why?
  - Leandro Aragoncillo, e.g.
    - Problem: Information release
    - Solution: Information Flow Control
- Stronger enforcement than reference monitors

Label and monitor

- Key:
  - tag data
  - monitor flows
- RMs tag actual data
  - all data/processes have label
  - central security monitor checks operations, data access against policy
- Security-typed languages use virtual tags
  - data types are labeled
  - type checker validates flows

Label all data

(CNN)

Monitor flows

Build on type safety

PENNSTATE 1855

- A type-safe language maintains the semantics of types. E.g. can't add int's to Object's.
- Type-safety is compositional. A function promises to maintain type safety.

Example 1

```
Object obj;
int i;
obj = obj + i;
```

X

Example 2

```
String proc_obj(Object o);
...
main()
{
  Object obj;
  String s = proc_obj(obj);
  ...
}
```

Labeling types

PENNSTATE 1855

Example 1

```
int{high} h1,h2;
int{low} l;
l = 5;
h2 = l;
h1 = h2 + 10;
l = h2 + l;
```

- Key insight: label types with security levels
- Security-typing is compositional

Example 2

```
String{low}
proc_obj(Object{high} o);
...
main()
{
  Object{high} obj;
  String{low} s;
  s = proc_obj(obj);
  ...
}
```

Implicit flows

PENNSTATE 1855

Static (virtual) tagging

```
int_Low mydata = 0;
int_Low mydata2 = 0;
if (test_High)
    mydata = 1;
else
    mydata = 2;
mydata2 = 0;
print_Low(mydata2);
print_Low(mydata);
```

`mydata` contains information about `test` so it can no longer be `Low`, but `mydata2` is outside the conditional, so it is untainted by `test`

Causes type error at compile-time

Declassification

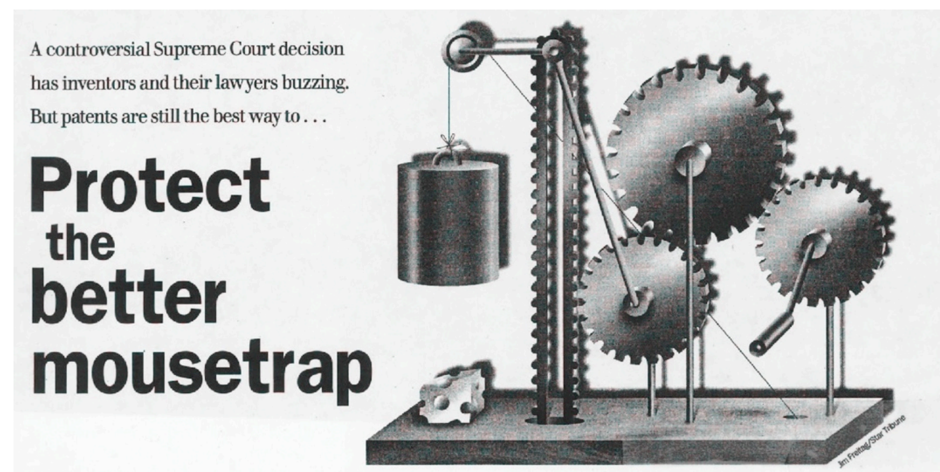
PENNSTATE
1 8 5 5

- MLS is too restrictive
- Examples:
  - Encryption
  - Distributed auction
  - Password check
- Solutions:
  - Declassification
    - Reduce the level of data -- tolerable leakage

Open challenges

PENNSTATE 1855

- System-wide security
- Certifying compilation
- Abstraction-violating attacks
- Dynamic policies
- Practical issues
- Variations of static analysis



Take away

PENNSTATE

1855

"The inability to express or enforce end-to-end security policies is a serious problem with our current computing infrastructure, and language-based techniques appear to be essential to any solution to this problem."