

RCDs and Secure Development

Refactored Cognitive Dimensions
and Secure Development

Luke Church
luke@church.name



'The vast majority of security failures occur at the level of implementation detail'

R. Morris, Senior NSA Scientist,
Quoted in Why Cryptosystems Fail



Agenda

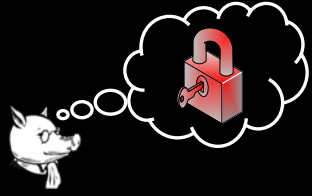
- What?
- Introductory Problems
- Old and New Languages
- Abstractions
- Other Psychology Issues
- Applications
- Discussion



What?

- **Premise:**

**Code Level Security Bugs are
a Psychology of Programming
Issue**



Trivial Model

- All security bugs are **Hidden Dependencies**
- -> The attacker can modify the behaviour of the system in an unexpected and harmful manner
- But this is not very useful...

Introductory Problems

Hidden Dependencies



Hidden Dependencies

- Dependencies on:
 - Semantics
 - Input
 - Syntax of interacting system (e.g. protocols)
- There are ***LOTS*** of them



Hidden Semantic Dependencies

- **Library function problems**
 - **Strcpy**
 - Developer thinks one behaviour
 - String -> String
 - Library implements it, with caveats
 - Data block -> Data block
 - **Hidden Semantic Dependency**
 - Premature commitment to length of buffer
 - Failed abstraction, Cognitive Mismatch
 - » Gave the impression of one thing
 - » Then did another

Languages

C++ and Java 1.5



The Old World

- C++
 - Take the hidden dependencies of a computer
 - Add some more
 - Give them to the developer
- Technical onslaught, much hope has been placed in 'new languages'
 - 'Security experts will be redundant with .NET'



Managed Languages

- **Java, C#**
 - Run on a Virtual Machine
 - No buffer over-runs
 - Security management by the run-time
 - Generally offers slightly fewer *Hidden Semantic Dependencies* and better *Domain Correspondence*
 - But have the fundamentals changed?



Can Syntactic Sugar cause your code to decay?

- **Case Study: Java's Generics**
- **Before Generics**
 - Untyped list
 - All items in list need to be cast

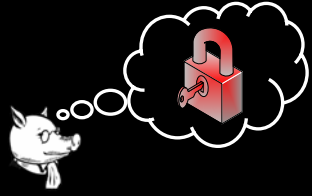
```
List cats = new ArrayList();  
cats.add(cat1); //OK  
cats.add(dog1); //OK  
Cat cat = (Cat)cats.get(0); //OK  
cat = (Cat)cats.get(1);  
//Runtime cast exception, it was a dog
```



Generic Version

```
List<Cat> cats = new ArrayList<Cat>();  
cats.add(cat1);  
cats.add(dog1); //Compile time error  
...  
Cat cat = cats.get(1);
```

- Conceptually: It's a list of cats 😊



Generic Sample 2

- What's wrong with this:

```
for (int i = 0; i < cats.size(); i++)
{
    Cat felix = cats.get(i);
    System.out.println(felix.getName());
    System.out.println(felix.getCall());
}
```

- Nothing...



How subtle does it get?

- Java does type erasure on generic types
`List<Cat> -> List<Object>`
- Compiler verification is used for much of the type checking
`List<Cat>.Add(Cat) -> List.Add(Cat)`
- The type parameter is 'erased'
- Dubious expressions give compiler warnings
- But, the **internal representation is a non-generic List**
- Reflection/Serialization/Dynamic Byte Code emission can probe this internal representation
- So I can add a Dog to a list of Cats!
- It only breaks when I try to remove the Dog from the list, potentially days later
- Then the system tears down with an unhandled exception
 - Denial of Service
- So `List<Cat>` cannot be trusted to only contain Cats!



What went wrong

- **Hidden semantic dependency**
 - Failed abstraction, Cognitive Mismatch
 - Gave the impression of one thing
 - Then did another
- Copy + Paste from slide on C++?
- Is Java really that different?

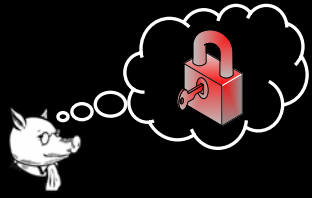


It gets worse

- **Compiling a working Java 1.4 program against Java 1.5 produces**
 - **Thousands of spurious Generics warnings**
 - You can't even use non-generics safely
 - **Generics isn't type safe, and creates a huge visibility problem**
- **The fundamentals aren't changing.**

**Abstractions and other
issues**

The effect of abstractions
on usability and other
psychological issues



Abstraction considered harmful?

- Yes, when it's not understood
 - And most of them aren't by most developers
- Make it simple
- Or don't make it at all
- But don't give us assembler and expect us to write secure code!



Other Psychological Issues

- **Motivation**
 - Security Issue
 - Testing failure pathways
- **Inverted Input-Space Problem**
 - Security often done by saying 'I don't want this to happen'
 - Programming Satan's computer
 - Error handling
 - Concurrency control



Cognitive Viscosity

- Security features are often unused
 - Security is a form of *Cognitive Viscosity*
 - Developer's idea of a task:
 - Query a database
 - Secure Developer's view
 - Ensure querying is safe
 - Query a database
 - Ensure data is safe to return
 - 1 task became 2 or 3 => **Cognitive Viscosity**

Applications



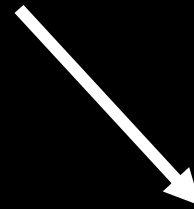
Design Methodology

- **Hidden dependencies**
 - Can it be removed?
 - Can it be made explicit?
 - Visibility everywhere 😊
- **Premature commitment**
 - Is it helpful?
 - Yes and No
- **Invert the Input Space (Again)**



Sample

```
Class UserName
{
string name;
    public UserName(string name)
    { this.name = name; }
//...
}
```



```
Class UserName
{
string name;
    [AllowInput("name", "[A-Za-z0-9]{5,10}")]
    public UserName(string name)
    { this.name = name; }
//...
}
```



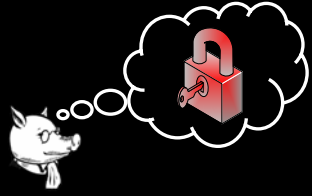
Where to go from here?

- So we have some beginnings of a model
 - Possible problems
 - Possible design manoeuvres
- How do we test it?

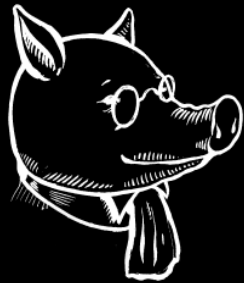


Conclusions

- Code level security issues are at least partially a cognitive problem
- Hidden semantic dependencies are the cause of many evils
- Not all that much has changed with modern languages
- Maybe we can do better with a Psychology of Programming understanding?



Discussion?



luke@church.name