

**REVERSE ENGINEERING
VULNERABILITIES IN SIMULATION
SOFTWARE**

Eric S. Imsand

Adam C. Sachitano

John A. Hamilton, Jr., Ph.D.

**Department of Computer Science and Software
Engineering
Auburn University**

Background and Goals

- Background
 - Ballistic Missile Defense
 - Defense simulation packages
- Goals
 - Sensitive information in simulation package (released form)
 - Platforms: Solaris and Windows

Four attack phases

- Run-time vulnerabilities
- Reverse translation
 - Disassembly
 - De-compilation
- Object File analysis
- Released Source Code analysis

Run-time vulnerabilities

- Ability to seize control of the executable
- Buffer overflow attacks, et. al.
- A vulnerability found, BUT:
- What can possibly be gained?

Reverse Translation – De-compilation

- Windows: DCC, DisC, REC
 - Immature
 - DCC & DisC targeted at specific compilers
- Solaris: REC
 - Compiler support too broad
 - Doesn't generate valid code on trivial programs
- Overall:
 - Unreadable, invalid code

Reverse Translation - Disassembly

- Windows: Hackman, PE Explorer
 - ~500 MB of assembly code with auxiliary info
- Solaris: 'dis'
 - ~100 MB executables => ~450 MB assembly
- Overall:
 - About 9.3 million lines of assembly generated

Object File Analysis

- String Literal Analysis
 - Debugging symbols found (~160000)
 - ~32000 format strings (printf, et. al.)
- Debuggers easier to use on ‘unstripped’ object files
- Stripping removes all possible debugging symbols

Included Source Code Analysis

- Comments left Intact!
 - i.e. “Capability x added on date y to support weapon platform z.”
- Calls to another sensitive simulator found
- No defined constants, variables, etc.
 - This is expected in a good, parameterized simulation

Conclusions

- Releasing unchecked source code is bad!
- Releasing unstripped binaries makes it easier to reverse-engineer with a debugger
- A dedicated adversary could still analyze by either method of reverse translation.
- A well-designed simulation package should be extensively parameterized