GRAMMATECH

U.S. AIR FORCE

PROTEUS

Autonomous Vulnerability Discovery and Remediation for Binary Code

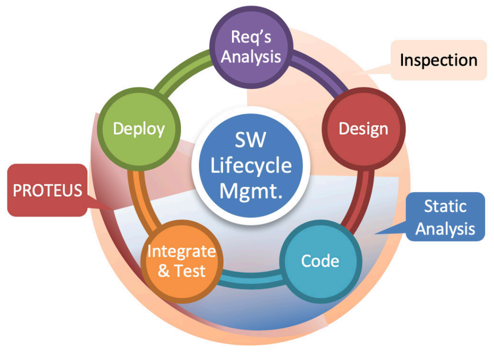
Static Application Security Testing (Binary)
Software Composition Analysis
Dynamic Application Security Testing
Interactive Application Security Testing
Runtime Application Self-Protection

Proof of Vulnerability

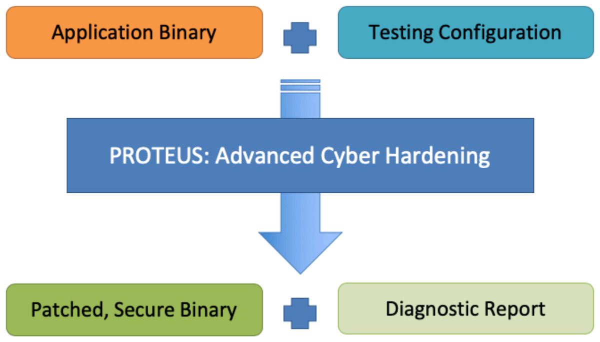
Design
✓ **Development**
Pre-Deployment
Deployment
✓ **Sustainment**

Problem	Applications often have unknown security vulnerabilities, which need to be discovered and then eliminated as soon as possible. Depending on vendors to address issues is not always sufficient: in the best case they will take time to develop new releases, in the worst case they will not address the issues at all.
Market Need	Detect memory corruption vulnerabilities in native binaries and offer rapid remediation.
Approach	Discover potential vulnerabilities; recommend and apply patches; develop security policies; harden executable against residual, undiscovered vulnerabilities.
Applications	Independent verification and validation of 3rd-party software; rapid patch deployment without waiting for or requiring vendor support; hardening of software binaries against memory-corruption exploits.

Vision



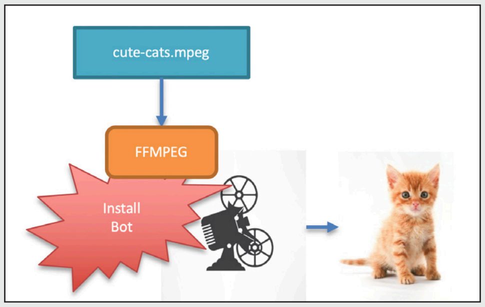
Improve security during the software development life cycle



Improve security during deployment

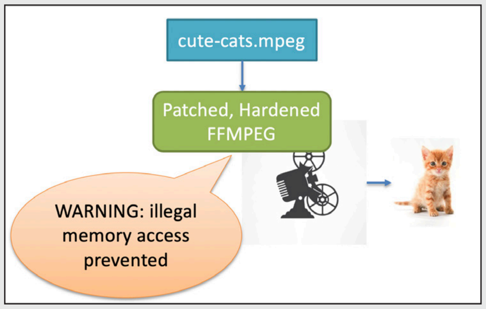
Motivating Example: FFMPEG (free movie player)

Security concern:



- Attacker crafts a malicious "mpeg"
- When run, cons FFMPEG into performing malicious actions, e.g., installing a bot.
- Publishes: "Cute cats video!!!".

Proteus-repaired, hardened FFMEG more resilient.



On attack, may:
- Safely continue execution
- Log the attempted attack
- Exit (cleanly)

Focus on Memory Corruption Vulnerabilities

CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
CWE-121: Stack-based Buffer Overflow
CWE-122: Heap-based Buffer Overflow
CWE-124: Buffer Underwrite
CWE-126: Buffer Over-read
CWE-127: Buffer Under-read
CWE-129: Improper Validation of Array Index
CWE-134: Uncontrolled Format String
CWE-170: Improper Null Termination
CWE-415: Double Free
CWE-416: Use After Free
CWE-457: Use of Uninitialized Variable
CWE-590: Free of Memory not on the Heap
CWE-665: Improper Initialization
CWE-761: Free of Pointer not at Start of Buffer
CWE-762: Mismatched Memory Management Routines
CWE-805: Buffer Access with Incorrect Length Value
CWE-806: Buffer Access Using Size of Source Buffer
CWE-824: Access of Uninitialized Pointer
CWE-908: Use of Uninitialized Resource

GRAMMATECH

FOR MORE INFORMATION
www.grammatech.com

EMAIL research@grammatech.com

© GrammaTech, Inc. All rights reserved.
GrammaTech Inc. Proprietary Information.

GRAMMATECH

U.S. AIR FORCE

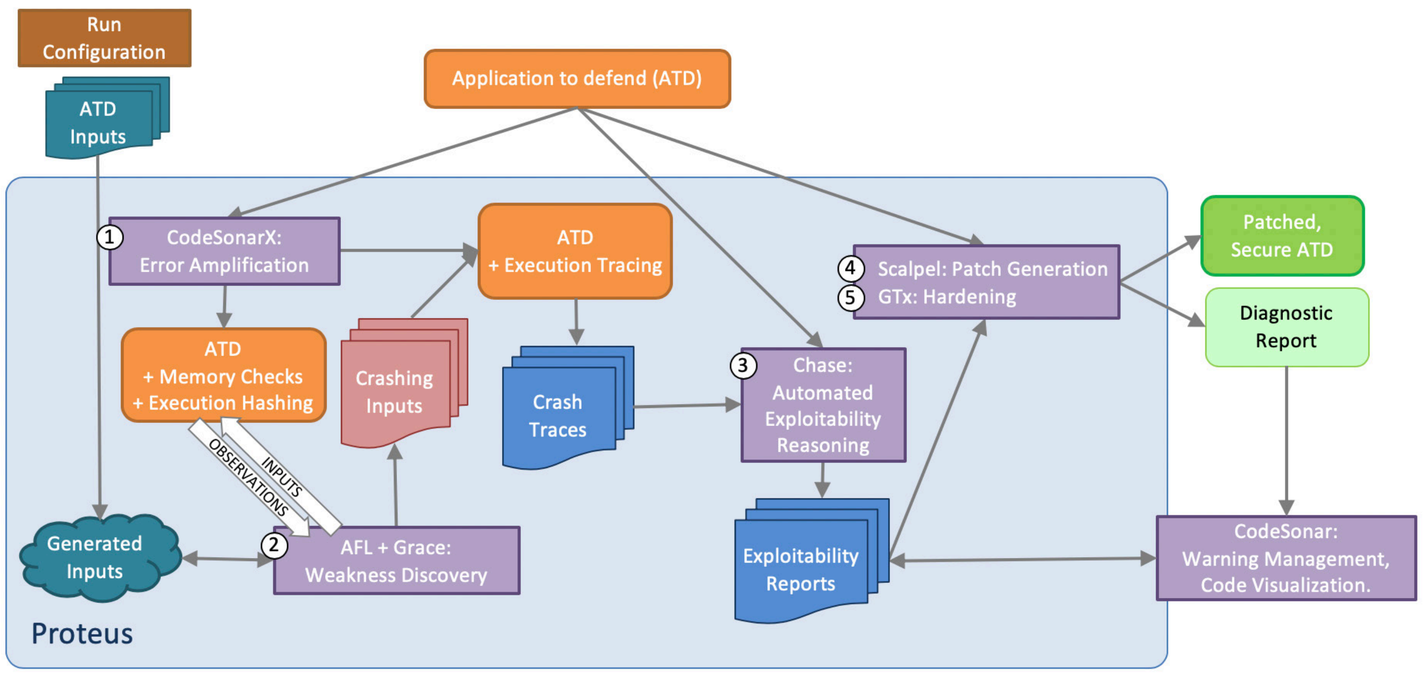
PROTEUS

Autonomous Vulnerability Discovery and Remediation for Binary Code

Static Application Security Testing (Binary)
Software Composition Analysis
Dynamic Application Security Testing
Interactive Application Security Testing
Runtime Application Self-Protection

Proof of Vulnerability

Design
✓ **Development**
Pre-Deployment
Deployment
✓ **Sustainment**



Run Configuration
ATD Inputs
Application to defend (ATD)
1 CodeSonarX: Error Amplification
ATD + Execution Tracing
4 Scalpel: Patch Generation
5 GTx: Hardening
Patched, Secure ATD
Diagnostic Report
ATD + Memory Checks + Execution Hashing
Crashing Inputs
Crash Traces
3 Chase: Automated Exploitability Reasoning
OBSERVATIONS
INPUTS
Generated Inputs
2 AFL + Grace: Weakness Discovery
Exploitability Reports
CodeSonar: Warning Management, Code Visualization.
Proteus

(1) Error Amplification

GrammaTech CodeSonarX

The goal of error amplification is to make noise early, which increases the rate of error detection and helps with fault localization. This goal is achieved by adding memory-safety monitoring instrumentation (like asserts) to the Application To Defend (ATD). A second goal is to help with analysis of any program crashes that are detected, which is done by instrumenting the ATD to capture an execution trace.

(2) Weakness Discovery

GrammaTech Grace
American Fuzzy Lop (AFL) Fuzzer

Grace is a symbolic execution engine that employs an SMT solver to generate program inputs that will exercise paths of interest.

Proteus uses Grace together with AFL (`http://lcamtuf.coredump.cx/afl/`) to discover crashing inputs that may indicate a potential vulnerability in the ATD.

3) Exploitability Analysis

GrammaTech Chase

Chase examines the program state after a crash to determine the underlying cause and its exploitability. For example, a crash due to a null pointer dereference is considered to have a low risk for exploitability, while a crash involving a control-flow transfer to an unmapped address range is judged as high risk.

(4) Binary Patching

GrammaTech Scalpel

Patches are applied to remove weaknesses that were determined to have a high likelihood of exploitability.

(5) Binary Hardening

GrammaTech GTx
GrammaTech Twitcher

GTx is a binary transformation tool. Given a binary executable, GTx transforms it to include extra instrumentation, change or drop functionality, or perform other user-specified transformations.

The hardening phase adds residual protections for any weaknesses that may remain undiscovered in the ATD.

(6) Validation (Regression Testing)

Proteus provides evidence explaining its conclusions and decisions, which empowers users to evaluate both the original ATD and the protected version created by Proteus.

Warning Management

GrammaTech CodeSonar®

CodeSonar® is GrammaTech's flagship static analysis SAST tool. It detects and explains weaknesses in source and binary code, and provides features for managing warnings (including warnings produced by other tools), visualizing code structure, customizing analyses, and more.

`https://www.grammatech.com/products/codesonar`

GRAMMATECH

FOR MORE INFORMATION
www.grammatech.com

EMAIL research@grammatech.com

© GrammaTech, Inc. All rights reserved.
GrammaTech Inc. Proprietary Information.