Grey-Box Penetration Testing for Full Life-Cycle Cybersecurity

Vector Security Symposium 2021, Dr. Christof Ebert
Agenda

- Welcome
  - Security Testing Methods
  - Security Testing in Practice
  - Summary and Discussion
Welcome

Vector Group

Employees:
> 3,300 Vectorians in 31 locations in 14 countries

Turnover:
692 Mio. € in 2020

Standardization,
Universities,
Publications

Customers:
> 9,000 companies in 76 countries

Portfolio:
- PLM/ALM
- Embedded SW
- Testing
- Diagnostics
- Calibration
- Consulting

Affiliated Companies:
GiN | CSM | BASELABS
Agenda

Welcome

- **Security Testing Methods**
  Security Testing in Practice
  Summary and Discussion
Security Testing Methods

Challenges with Security Implementation, Verification and Validation

- **Security needs a system perspective**
  - Cost efficiency with more frequent delivery cycles
  - Transparent and efficient regression strategy
  - Traceability and documentation for software update management systems (SUMS)
  - Tool chain from impact analysis to deployment

- **Security testing needs new methods**
  - Defensive coding, e.g., memory allocation, avoid injectable code, least privileges, pointers, zero page
  - High cryptographic strength which balance performance and life-cycle risks, e.g., quantum computing
  - Considering automatic functions with ML – and the related novel attack schemes, such as faking
  - Automatic regression tests for each delivery
  - Risk-based penetration testing
  - Awareness and governance towards social engineering

Classic coverage test is not sufficient anymore. Test for the known – and for the unknown.
Security Testing Methods

Security by Lifecycle: Verification, Validation and Life-Cycle Management

- **Code and Design Analysis**
  - Static: Code Quality Analysis (CQA), defect analysis
  - Dynamic: specific unit tests, coverage (e.g., VectorCAST)

- **Fuzz Testing**
  - Brute-force CAN Fuzzer for fuzzing the Application SW
  - Interface scanner, layered fuzzing, crackers, vulnerability scanner

- **Pen Testing**
  - Connect with misuse, abuse and confuse cases
  - Vector Grey-Box PenTest based on TARA and risks
  - DoS, Replay, Mutant/Generated Messages

- **Post-SOP and continuous maintenance**
  - PSIRT Collaboration (Product Security Incident Response Team)
  - Handover, task assignments and distribution
  - OTA and SUMS: Ensure that each deployment satisfies security requirements

- Data encryption: Protection of intellectual property by encryption
- Authorization: Protection against unauthorized ECU access
- Validation: Safeguarding of data integrity e.g., in the flash memory
- Authentication: Verification of authenticity through signature methods
- Governance: Safety/security documentation is continuously updated
Security Testing Methods

From TARA to Requirements, Design, Test, and Traceability

- **Requirements**
  - Assets, TARA, Security Goals
  - Functional security requirements
  - Technical security requirements

- **Architecture**
  - System
  - Functional
  - SW/HW

- **Test**
  - Grey-Box Penetration Test, Robustness Tests, Fuzzing
  - Functional Tests, Security Testing
  - Static Code Analysis, Unit Test,
Systematically Deploy 10 Steps for a Risk-Oriented Security Test

1. Market Requirements
2. Product Requirements
3. Re却nnaissance
4. Specifications
5. Component Requirements
6. Integration Testing
7. Minimum Viable Penetration Test Cases
8. Acceptance Testing
9. Key Performance Indicators
10. Regression Testing

Software Development:
- Underlying Concepts
- Security Testing: 10 steps
- Grey-Box PenTest

### Determine Risk Exposure with TARA Results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AST 7</td>
<td>CAN Communication</td>
<td>Integrity</td>
<td>Compromise bus communication and inject not allowed messages (Integrity)</td>
<td>False positive alarm signals are sent to CAN-Bus. Safety functions activated although not needed</td>
<td>8</td>
<td>ECU Safety functions can not be used due to heavy operational impact.</td>
<td>Layman</td>
<td>Low</td>
<td>Spec COTS</td>
<td>3</td>
<td>Pot Heavy</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>No effect</td>
<td>High</td>
<td>The ECU shall ensure message authentication and message integrity through CAN-Bus</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CAL (Risk value)</th>
<th>Attack Feasibility Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negligible</td>
<td>--- (1)</td>
</tr>
<tr>
<td>Moderate</td>
<td>CAL1 (1)</td>
</tr>
<tr>
<td>Major</td>
<td>CAL1 (1)</td>
</tr>
<tr>
<td>Severe</td>
<td>CAL2 (1)</td>
</tr>
</tbody>
</table>

**CAL** (Risk value)
Ensure Traceability with Triple Peak Model

**Security Testing Methods**

Forward TDRE: Test Cases are synthesized together with requirement for testability.

Backward TDRE: Test cases are executed, two cases are possible:
1. Positive result: Addition of new requirement
2. Negative result: Potentially redundant test case, invalid issue

---

Security Testing Methods

Efficiently Combine Test Strategies with Fuzzing and Penetration Testing

- Confirm findings of the Fuzz Test, e.g., Occurrence of hard resets
- Mount network-based attacks, e.g., Replay, Man in the Middle, Tampering of Data
- Read and Write into the flash dump, e.g., extract keys, certificates, sensitive data

The Vector Pen Test ambition is to check for the CIAAG properties, specifically Confidentiality, Integrity and Availability.

- Q. Can we read the key, flash?
- Q. Can we modify the flash or CAN messages?
- Q. Can we perform a DoS and make the service unavailable?

Result of Fuzz Test

Hard resets / crashes due to invalid value, node inactive, incrementing of reset counter

- Confirming Fuzz Test results
  Reproduction of test cases with help of Interactive Generator (IG) in CANoe
- Send out those particular frame IDs using IG Generator and confirm for reset
- Mount network-based attacks like DoS, Man in the middle, tampering of signals
- Access the debug ports like JTAG to extract + corrupt flash memory / firmware
- Flash dump – Look for certificates, keys, passwords, sensitive data
## Security Testing Methods

### Conclusion: Combine Different Methods and Use KPI for Effectiveness and Efficiency

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Manual Reviews</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Static Code Analysis</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Unit Testing</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>System Testing</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Fuzzing</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Side Channel Analysis</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>Penetration Test</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

**Grey-Box PenTest** = Black Box PenTest + Brute Force + Intelligent Analysis

<table>
<thead>
<tr>
<th>White Box PenTest</th>
<th>Black Box PenTest</th>
<th>Grey Box PenTest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proprietary design knowledge</td>
<td>Brute force, Public domain knowledge</td>
<td>Intelligent analysis, Experiences, heuristics</td>
</tr>
</tbody>
</table>
Agenda

Welcome

Security Testing Methods

• Security Testing in Practice

Summary and Discussion
Grey-Box Penetration Testing in 10 Steps: Start with the End in Mind

**Goal:** Understand usage and gather information about misuse and abuse cases.

**Approach:** Apply passive reconnaissance even without direct interaction with the product. Start with market requirements, customer-provided documents of the system, brochures, user manuals, published material on attacks and vulnerabilities of used components, etc.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Usage View, Misuse and Abuse Cases View: Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic feature</td>
<td>Displays the vital stats of the vehicle like engine temperature, engine speed, car speed, petrol level from the sensors on the dashboard</td>
</tr>
<tr>
<td>User goal</td>
<td>Provides an all-in-one view of indicating the vehicle health</td>
</tr>
<tr>
<td>Target group</td>
<td>Driver of the vehicle</td>
</tr>
</tbody>
</table>
| Misuse and abuse cases            | • Tampering of engine speed leading to unintended braking.  
                                    | • Tampering the petrol level leading to stopping of engine.  
                                    | • Unauthorized reprogramming of controller to corrupt the functionality  
                                    | • Tampering the car speed, indicating rash driving, hence increasing the insurance |
| Impact of cyberattack             | Driver distraction, leading to accidents due to unintended braking or engine killing, increase in insurance,             |
Security Testing in Practice

Understand and Evaluate Underlying Architecture

- **Goal:** Analyze high-level architecture and interfaces from product requirements, to understand interaction between different components.

- **Approach:** Architecture model, System Context, Interface model, Attack pathway, Sequence Diagram.

---

![Diagram of architecture and interface modelling with system context, components, and interfaces.]
**Goal:** Identify relevant assets, such as hardware, software, databases, security-related information, user data, proprietary information of supplier, etc.

**Approach:** Parse through every requirement elicit assets which contain value to stakeholders. Asset Elicitation: Safety, Finance, Operational Performance, Privacy (S,F,O,P) scheme. Trace every product requirement ID with Asset ID.

### Identify Valued and Protectable Assets

<table>
<thead>
<tr>
<th>Req. ID</th>
<th>Requirement Description</th>
<th>Asset</th>
<th>Asset ID</th>
<th>Category</th>
</tr>
</thead>
</table>
| /FR 1/  | When Car Speed > 300 kmph, glow the red light on the dashboard and start the buzzer. | • Car Speed CAN Signal  
• Buzzer Alert signal | /AST 1.1/  
/AST 1.2/ | Safety, Operational Performance |
| /FR 2/  | When there is a change in the petrol level, engine speed, car speed, engine temperature values, display the changed value on dashboard | • Petrol Level CAN Signal  
• Engine Speed CAN Signal  
• Car Speed CAN Signal  
• Engine Temperature CAN Signal | /AST 1.3/  
/AST 1.4/  
/AST 1.5/  
/AST 1.6/ | Safety, Operational Performance, Privacy |
| /FR 3/  | When a New Software Update is TRUE, Software Update signal is set to TRUE, Controller accepts update from remote programmer. | • New Software Update Wi-Fi packets  
• Software Update signal | /AST 1.7/  
/AST 1.8/ | Safety, Privacy |
Goal: Obtain knowledge on internal details of device in terms of hardware, software specifications, to know about typical attacks and common vulnerabilities of device.


<table>
<thead>
<tr>
<th>Specification</th>
<th>Known vulnerability as published in CVE database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux kernel 5.5.0</td>
<td>CVE 2018-6705: Privilege escalation to perform arbitrary commands</td>
</tr>
<tr>
<td>BLE 4.2</td>
<td>CVE 2020-11975: Predictable random number during pairing leads to MITM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attack Description</th>
<th>Vulnerability</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015: Black Hat USA: Exploited the bug in UMTS receiver code to get complete control of vehicle, display, audio, control steering wheel</td>
<td>Bugs in code for processing of UMTS commands, Lack of security controls for the Wi-Fi connection</td>
</tr>
<tr>
<td>2018: Subaru StarLink Head Unit: Security researcher injects malicious firmware via USB, gains root code access</td>
<td>Unauthorized rewriting of firmware by exploiting the update mechanism</td>
</tr>
</tbody>
</table>

Processor: Dual core ARM Cortex M3
Wireless comm.: RF, BLE 4.2, modulation: GFSK Op. Freq.: 2.4GHz

OS, version: Linux kernel 5.5.0, x64
COTS SWL IP Stack ID: OpenAPS software, TCP/IP: 6.0.1.66
Prioritize Risks with a Mini-TARA (Threat Analysis and Risk Assessment)

- **Goal:** Analyze damage scenarios and threats; determine risk levels for their attack pathways.
- **Approach:** Risk assessment based on asset-trade off analysis: Impact level, Attack feasibility level, Risk level. Top-down approach: Assets > Damage > Threat > Attack vectors > Risk level.

<table>
<thead>
<tr>
<th>ID</th>
<th>Asset</th>
<th>CIAAG</th>
<th>Damage Scenario</th>
<th>Threat Scenario</th>
<th>Attack Vector</th>
<th>TL</th>
<th>IL</th>
<th>Security Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>TARA 1</td>
<td>AST 1.1: Car Speed CAN Signal</td>
<td>Integrity</td>
<td>Unintended braking due to high car speed as indicated on dashboard</td>
<td>Spoof sensor and tampering readings causing unacceptably high car speed</td>
<td>Compromise RF / BLE / UMTS / Wi-Fi port of Head Unit ECU, tamper the specific CAN messages</td>
<td>High</td>
<td>Very High</td>
<td>5</td>
</tr>
<tr>
<td>TARA 2</td>
<td>AST 1.3: Petrol Level CAN Signal</td>
<td>Integrity</td>
<td>Unintended stopping of engine due to critically low fuel message on dashboard</td>
<td>Spoof sensor and tampering petrol level readings to show very low fuel levels</td>
<td>Compromise RF / BLE / UMTS / Wi-Fi port of Head Unit ECU, tamper the specific CAN messages</td>
<td>High</td>
<td>Very High</td>
<td>5</td>
</tr>
</tbody>
</table>

**Factors to determine Threat Level (TL):**
- Elapsed Time
- Specialist Expertise
- Knowledge of Target
- Window of Opportunity
- Equipment Effort

**Factors to determine Impact Level (IL):**
- Safety
- Finance
- Operational Performance
- Privacy
Goal: Generate minimum, complete and non-redundant penetration test cases repository to cover relevant vulnerabilities and to document on underlying expected coverage ranges

Approach: Trace from assets to vulnerabilities; identify minimum viable test cases by evaluating KPI such as test effort and focus on high-risk items included in test case repository.

<table>
<thead>
<tr>
<th>Assets</th>
<th>Damage Scenario</th>
<th>Threat Scenario</th>
<th>Attack Vectors</th>
<th>Risk Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car Speed CAN</td>
<td>Unintended braking due to high car speed on dashboard</td>
<td>Spoof sensor and tamper readings causing unacceptably high car speed.</td>
<td>Compromise RF / BLE / UMTS / Wi-Fi port of Head Unit ECU, tamper the specific CAN messages</td>
<td>5</td>
</tr>
<tr>
<td>Petrol Level CAN</td>
<td>Unintended stopping of engine due to critically low fuel message on dashboard</td>
<td>Spoof sensor and tampering petrol level readings to show very low levels</td>
<td>Compromise RF / BLE / UMTS / Wi-Fi port of Head Unit ECU, tamper the specific CAN messages</td>
<td>5</td>
</tr>
<tr>
<td>Software Update Signal</td>
<td>Software corruption in controller memory.</td>
<td>Spoof remote programmer and tamper controller software causing dashboard malfunction</td>
<td>Compromise RF / BLE / UMTS / Wi-Fi port of Head Unit ECU, tamper the specific CAN messages</td>
<td>5</td>
</tr>
<tr>
<td>Buzzer Alert signal</td>
<td>Customer distraction, battery drain due to repeated buzzer alerts.</td>
<td>Spoof controller and cause excessive buzzer alerts leading to battery drain, driver distraction.</td>
<td>Compromise RF / BLE / UMTS / Wi-Fi port of Head Unit ECU, tamper the specific CAN messages</td>
<td>3</td>
</tr>
<tr>
<td>Car Speed CAN, Engine Speed CAN</td>
<td>Increase in insurance due to erroneously high car, engine speed indicating rash driving</td>
<td>Spoof sensor and tamper readings causing unacceptably high car speed and engine speed.</td>
<td>Compromise RF / BLE / UMTS / Wi-Fi port of Head Unit ECU, tamper the specific CAN messages</td>
<td>2</td>
</tr>
</tbody>
</table>
Perform Risk-Based Grey-Box Penetration Testing

**Goal:** Re-create attack scenario on device to identify if potential vulnerabilities are valid or invalid.

**Approach:** Intelligent error seeding: Injecting vulnerabilities to measure effectiveness and efficiency. Penetration test performed using minimum viable test cases (attack vectors). Perform additional checks, e.g., SUMS with OTA etc.

- Connect to Communication Bus – Check for network responsiveness
- Create Signal, Frame Fuzz tests in vTESTstudio
- Execute Test cases in CANoe
- Mount network-based attacks Memory / extraction and corruption
- Perform Additional Checks – SW Integrity, authenticity checks

**Example: Flashing SW checks:**
- Invalid address
- Modified SW
- Odd address
- Invalid signature
- Invalid Seed-Key
- Downgrade software
- Secure boot functionality
Security Testing in Practice

Measure Test Effectiveness and Test Efficiency

▶ **Goal:** Optimize security testing performance and cost by using automatic KPI

▶ **Approach:** Apply the KPIs to determine test effectiveness, e.g., vulnerability-finding ability, and test efficiency, e.g., required test effort

**Test Effectiveness:**

**Vulnerability Category Coverage:** Number of high-risk vulnerabilities identified by test strategy. Vulnerability categories are derived from TARA, public vulnerability databases and heuristics they consist of the potential vulnerability categories. For instance: Insecure CAN communication, Insecure Wi-Fi port.

\[
\text{No. of vulnerability categories discovered} \div \text{Total no. of vulnerability categories}
\]

- If a test strategy covers 8 out of 13 vulnerability categories, then it is 62% effective in identifying high-risk vulnerabilities.
- If it covers all vulnerability categories, hence it is 100% effective.

**Test Efficiency:**

**Vulnerability Discovery Average:** Average number of vulnerabilities found per test case.

\[
\text{No. of vulnerability categories discovered} \div \text{Total no. of test cases}
\]

- If a test strategy needs 1 test case to identify 1 vulnerability, then it is 100% efficient.
- If it requires 5 test cases to identify 1 vulnerability, then it is 86% efficient.
Iteratively Update Functional Security Requirements

**Goal:** Update security requirements to address vulnerabilities discovered, to perform iterative testing to ensure all new and existing vulnerabilities are resolved.

**Approach:** Based on vulnerabilities found, enhance requirements accordingly. Perform Grey-Box Pen Test once again starting from step 3 product requirements till end.

Continuous update functional security requirements along the life-cycle:

- Systems and service engineering methods for embedded and IT
- Scalable techniques for design, upgrades, regressions, services
- Multiple modes of operation (normal, attack, emergency, etc.)

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Status</th>
<th>New Security Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insecure Wi-Fi port: Infotainment Head Unit</td>
<td>Valid</td>
<td>Implementation of mutual identity authentication and Advanced Encryption Standard mechanism.</td>
</tr>
<tr>
<td>Insecure CAN communication: Car Speed CAN signal</td>
<td>Valid</td>
<td>Implementation of SecOC protection on this Car Speed CAN signal.</td>
</tr>
</tbody>
</table>
Cover the ENTIRE Life-Cycle with Security Testing

- **Goal:** Establish and maintain an efficient yet systematic regression testing to ensure all new and existing vulnerabilities are resolved.

- **Approach:** Perform impact analysis based on new requirements vulnerabilities found. Perform Grey-Box Pen Test as a semi-automatic process starting from step 3 product requirements till end.

**Regression Testing:**
- Existing functionality cannot be impacted.
- New and changed security implementation is working as expected.
- Identification of new threats and vulnerabilities due to requirement changes.
- Iterate steps 2 to step 9 and conclude regression cycles when valid vulnerabilities tend to 0.
- Clean up scripts, executables, temporary files: Reconfigure settings back to original state.
Agenda

- Welcome
- Security Testing Methods
- Security Testing in Practice
  - Summary and Discussion
Vector Offers the Most Complete Portfolio for Cybersecurity

Vector Cybersecurity Solutions

Consulting and services
• SecurityCheck and SafetyCheck
• TARA
• Security concept
• Code analysis
• PenTesting
• Virtual Security Manager

Tools
• COMPASS SecurityCheck and TARA
• VectorCAST for code analysis and coverage
• Security Manager Extension for Vector Tools and Fuzz Testing
• PLM with PREEvision
• Diagnosis

AUTOSAR Basic Software

HSM for HW based Security

Engineering Services for Security

www.vector.com/security
Thank you for your attention.
Please contact us for consulting support.


Vector Consulting Services

@VectorVCS
www.vector.com/consulting
consulting-info@vector.com
Phone: +49-711-80670-1520