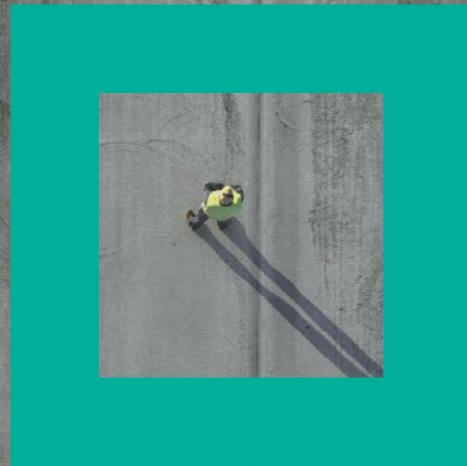


Forcepoint Product & Security Testing Overview

Paul Hemson - VP of Quality

James Fudge - Director of Software Engineering

Forcepoint



Agenda

- Test Strategy
 - What, Why, How
- Application Security Testing
 - Forcepoint's Product Security Services
 - Secure Development Lifecycle
 - Security Test Strategy
 - Threat Modeling & Case Study
 - Penetration Testing
 - Dynamic Analysis
 - Static Analysis
 - Fuzzing
 - Vulnerabilities & CVSS
 - References

Forcepoint Overview

- A human-centric approach to cyber security
 - Humans are the new perimeter
 - Understand and respond to Indicators of Behavior (IOB's)
 - Approach has been validated during Covid-19
- The company
 - ~2000 people
 - ~800 in Engineering
 - Grew through acquisitions and mergers of 5 companies over past 5-6 years

The Test Strategy

- What it is
 - A defensible description of what will and will not be tested, and why
- Why it is important
 - There is limited time and effort available to mitigate quality risks that are important to customers
 - Risks must be identified and socialized
 - It's easy to spend (waste) time & effort on work which adds little or no value

Test Strategy – Some details

- Project specific
 - Large & small
 - Use templates, checklists & examples – but avoid “boilerplate”
- Makes the (testing) work visible
 - The work that is done
 - The work that is not done
- Must be defensible
 - Internally, with exec’s and other functional groups
 - Externally, in both proactive and reactive discussions
- Must be adaptable
 - Between releases & within releases
- A basis for discussion
 - Why, what, how, how much, when, why not...

Testing Strategy - Example

A template within Confluence used for planning and high-level tracking of execution

Planning

- What tests will provide us with the information we need to feel confident about this release?
- How much effort to spend on it?

Execution

- Are we executing on our plan?
- What new information (bugs) are we finding based on the plan?

Release Candidate/Go-NoGo


- Have we completed our plan?
- How accurate were effort estimations?
- Where did we spend the most effort?
- What testing activities proved most useful (measured by # of bugs)?

Testing Activities to Consider

	Testing Type	Description - Edit each row and describe the planned activities and targeted areas	Status	Jira Story and SubTask Links	Planned Effort	Actual Effort	Jira Bug Links
1	Unit	Implemented by the person who wrote the code using white box testing techniques and reviewed by an independent peer	PLANNED				
2	Automation	Automate what needs to be automated, i.e. anything that cannot be tested manually or it would be a pain to do so	PLANNED				
3	Integration	To ensure that the coupling between various software modules is working as intended with no adverse impact, endpoint products are functioning correctly with DLP Server, DLP classifier, FIT, ECA cloud, and DCEP/PCEP cloud environment	DEFINED				
4	Functional	To ensure that all new functionality satisfies acceptance criteria	COMPLETE				
5	Use Case/End-to-End	Ensure all use cases for the product, release, and solution are working as expected	DEFINED				
6	Currency	To ensure that the endpoint products work as expected in all intended target environments	BLOCKED				
7	Acceptance	Cursor testing to ensure that the components of endpoint product builds are functioning at the basic level and further in-depth testing can continue	COMPLETE				
8	Stress/Stability	Running for 6 days continuously without system crashes, memory and handle leaks under various levels of simulated usage	DEFINED				
9	Performance	Ensure the performance requirements and customer expectations regarding performance of the endpoint products are met	DEFINED				
10	Security	Ensure the product has no security vulnerabilities	OUT OF SCOPE				
11	Exploratory	Creative, non-scripted, testing within a focused area of the product	DEFINED				
12	Regression	Ensure old functionality still works as expected after code changes have been made	COMPLETE				
13	Install/Uninstall	Ensure products install and uninstall as expected	COMPLETE				
14	Upgrade/Downgrade	Ensure products can be upgraded and downgraded as expected	COMPLETE				

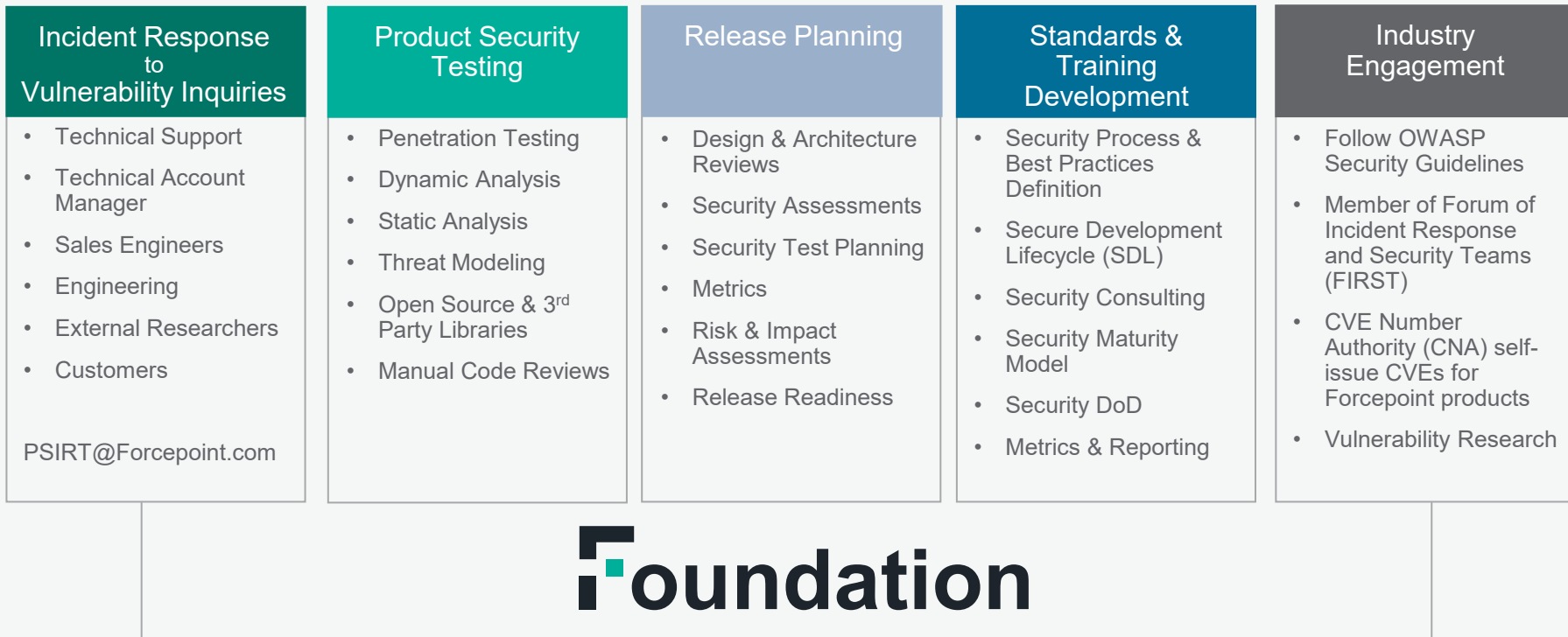
Application Security Testing Agenda

- Forcepoint's Product Security Services
- Secure Development Lifecycle
- Security Test Strategy
- Threat Modeling & Case Study
- Penetration Testing
- Dynamic Analysis
- Static Analysis
- Fuzzing
- Vulnerabilities & CVSS
- References



Security doesn't
happen by accident,
and it is everybody's
responsibility

Product Security Services



Secure Software Development Lifecycle Program

- Security Definition of Done
- Secure Architecture Design Reviews
- Threat Modeling
- Secure Coding Standards
- Manual Code Reviews
- Static Analysis 3rd Party & Open Source
- Static Analysis (Company Code)
- Dynamic Analysis
- Penetration Testing
- Vulnerability Management Reporting

Secure Software Development Lifecycle Program

Created by James Fudge just a moment ago

SSDLC Activities

Security Definition of Done

- Owned by: Product Security Team
- Applied and enforced on each product: Security Champions, Program and Project Manage
- Methodology: Based on Product Security Definition of Done
- Tools: Security Definition of Done checklist
- Requirement: Mandatory for every project
- Deliverable: DoD artifact of compliance

Secure Architecture Design Reviews

- Owned by: Security Champions
- Methodology: Security Architecture Design Review Process
- Requirement: Mandatory for every project
- Deliverable: SADR Checklist artifact of compliance

Threat Modeling

- Owned by: Security Champions
- Methodology: PST STRIDE Model (Microsoft)
- Tools: InsiRisk, MS STRIDE Model
- Requirement: Mandatory for every project
- Deliverable: Threat model artifact document

Secure Coding Standards

- Owned by: Product Security Team
- Applied and enforced on each product: Security Champions
- Recommendations:
 - Carnegie Mellon University Software Engineering Institute Secure Coding Practices
 - Open Web Application Security Project Secure Coding Practices
- Requirement: Mandatory for every project
- Deliverable: Security Champion determines what items or subset of the standards apply to

Manual Code Reviews

- Owned by: Product Team
- Applied and enforced on each product: Security Champions
- Methodology: Follow these 9 Best Practices for manual code reviews while reviewing code
- Requirement: Mandatory for every project
- Deliverable: Report of any variance based on the coding standards that are selected and applied

Static Analysis Open Source & Third-Party Libraries

- Owned by: Security Champions
- Methodology: Automatic scanning
- Tools: [Ifrog Gray](#)
- Requirement: Mandatory for every project
- Deliverable: Artifact of scan results, Jira dashboard and report of findings

Static Analysis Forgepoint Code

- Owned by: Security Champions
- Methodology: Automatic scanning
- Tools: [SonarQube](#), [Checkmarx](#)
- Requirements: Mandatory for every project
- Deliverable: Artifact of scan results, Jira dashboard and report of findings

Dynamic Analysis Fuzz Testing

- Owned by: Security Champions
- Methodology: Each team identifies and develops product specific tests that are applied to a given release
- Tools: Each team develops a specific fuzzing tool based off of a standard framework developed by the Product Security Team. [OWASP Fuzzing reference](#).
- Requirement: Mandatory for every MDR project initiated after 1-January 2021
- Deliverable: Artifact of fuzzing results, Jira dashboard and report of findings

System Security Testing (required of the product team)

- Owned by: Security Champions
- Methodology:
 - Identify how any new or changed feature or capability affects the product's attack surface and threat boundary.
 - Refer to the Threat Modeling section above for details on those expectations.
 - Design and execute appropriate tests to exercise the relevant functionality, ensuring that no additional security issues have been introduced
- Requirement:
 - Mandatory for every project
- Deliverable:
 - Documented overview of the change, testing performed, test completion status and results of that testing
 - Review results and findings with the Product Security Champion, Engineering Leader, and Product Owner

Penetration Testing

- Owned by: Product Security Team
- Methodology: Based on the Product Security Assessment Strategy Template
- Tools: Nessus, Qualys, HCL [Appscan](#), [BurpSuite](#) Professional, [SSLScan](#) etc.
- Requirement:
 - Mandatory for every project
- Deliverable: Penetration Testing Security Assessment Findings Report

Reporting Security Vulnerability Management (SVM)

- Owned by: Product Security Team (PST)
- Methodology: Product security assessments
- Tools: SVM [Jira](#) Creation executable
- Deliverable: Jira SVM Dashboards

Security Test Strategy

Security Testing Activities to Consider

Ref No.	Testing Activity Type	Description - Edit each row and describe the planned activities and targeted areas
1	Scope Request Assessment	Review the scope of the planned assessment, this usually includes a review of the initiatives, and a walk-through with team Program/Project Manager
2	Architecture Review	Review the planned assessment architecture changes/updates with the development team and architects
3	Define Assessment Scope	Define the scope of the engagement (what will be covered & who will be doing what), this activity feeds into the actual assessment test plan
4	Threat Modeling	Threat model the product to help determine potential security threats and vulnerabilities, this activity feed into the security assessment test plan
5	Static Analysis	Execute SonarQube on source code to look for potential product vulnerabilities or security concerns such as XSS, Buffer etc.
6	Dynamic Analysis	Execute dynamic analysis testing (HCL Appscan) to look for potential product vulnerabilities or security concerns
7	Open Source & 3rd Party Lib evaluation	Execute JFrog Xray , GitHub Dependabot or equivalent tool against source code to look for out-of-date, non-supported security concerns
8	Manual Security Code Reviews	Conduct manual code reviews on security related areas to look for potential product vulnerabilities or security concern read-only source code access)
9	Fuzz Testing	Execute fuzz testing (automate the input of random data to input fields), to look for system errors, or security concerns

11	SVM Review	Review list of outstanding SVMs and see how they align against the vulnerability remediation SLAs
12	Definition of Done	Review security assessment DoD, used as part of the go/no-go decision
13	Security Maturity Model	Provide a "Security Maturity Model" rating (based on the 1-4 rating system)
14	Technical Risk & Impact Statement	Provide technical risk and impact statement (could be combined with assessment report)
15	Assessment Report	Deliver Product Security Assessment report (will be used in go/no-go meeting)
16	Engagement Retrospective	Product Security Assessment engagement retrospective with PST and product team following a stop/start/continue meeting

10	Penetration Testing	Execute pen testing on system, network infrastructure or Web app (this may include some of the other testing activities in this table)												
10.a	System/Network Infrastructure Pentest	<div>Internally Developed: System/Network Infrastructure Pentest Checklist</div> <table><tr><td>System/Network Infrastructure Pentest</td></tr><tr><td>Host Discovery Network Mapping</td></tr><tr><td>Non-Credentialed Vulnerability Assessment</td></tr><tr><td>Credentialed Vulnerability Assessment</td></tr><tr><td>Assess SSL/TLS & Cipher Suites</td></tr><tr><td>Review Running Processes for Use of Deprecated Software/Libraries</td></tr><tr><td>Review Python/Perl/JavaScript Usage for Deprecated Modules</td></tr><tr><td>Review Processes running as root user</td></tr><tr><td>Authentication, Authorization, and Accounting Between Components</td></tr><tr><td>Review of System Log Management</td></tr><tr><td>Review of File Privileges</td></tr></table>	System/Network Infrastructure Pentest	Host Discovery Network Mapping	Non-Credentialed Vulnerability Assessment	Credentialed Vulnerability Assessment	Assess SSL/TLS & Cipher Suites	Review Running Processes for Use of Deprecated Software/Libraries	Review Python/Perl/JavaScript Usage for Deprecated Modules	Review Processes running as root user	Authentication, Authorization, and Accounting Between Components	Review of System Log Management	Review of File Privileges	
System/Network Infrastructure Pentest														
Host Discovery Network Mapping														
Non-Credentialed Vulnerability Assessment														
Credentialed Vulnerability Assessment														
Assess SSL/TLS & Cipher Suites														
Review Running Processes for Use of Deprecated Software/Libraries														
Review Python/Perl/JavaScript Usage for Deprecated Modules														
Review Processes running as root user														
Authentication, Authorization, and Accounting Between Components														
Review of System Log Management														
Review of File Privileges														
10.b	Web Application Pentest	<div>From https://owasp.org/www-project-web-security-testing-guide/</div> <table><tr><td>Web Application Pentest</td></tr><tr><td>Configuration and Deployment Management Testing</td></tr><tr><td>Identity Management Testing</td></tr><tr><td>Authentication Testing</td></tr><tr><td>Authorization Testing</td></tr><tr><td>Session Management Testing</td></tr><tr><td>Input Validation Testing</td></tr><tr><td>Testing for Error Handling</td></tr><tr><td>Testing for weak Cryptography</td></tr><tr><td>Business Logic Testing</td></tr><tr><td>Client Side Testing</td></tr><tr><td>WebSockets*</td></tr></table> <div>*https://kennel209.gitbooks.io/owasp-testing-guide-v4/content/en/web_application_security_testing/testing_websockets_client-010.html</div>	Web Application Pentest	Configuration and Deployment Management Testing	Identity Management Testing	Authentication Testing	Authorization Testing	Session Management Testing	Input Validation Testing	Testing for Error Handling	Testing for weak Cryptography	Business Logic Testing	Client Side Testing	WebSockets*
Web Application Pentest														
Configuration and Deployment Management Testing														
Identity Management Testing														
Authentication Testing														
Authorization Testing														
Session Management Testing														
Input Validation Testing														
Testing for Error Handling														
Testing for weak Cryptography														
Business Logic Testing														
Client Side Testing														
WebSockets*														
10.c	API Pentest	<div>From https://owasp.org/www-project-api-security/</div> <table><tr><td>API Pentest</td></tr><tr><td>Broken Object Level Authorization</td></tr><tr><td>Broken User Authentication</td></tr></table>	API Pentest	Broken Object Level Authorization	Broken User Authentication									
API Pentest														
Broken Object Level Authorization														
Broken User Authentication														

Threat Modeling

- Introduction

- Threat modeling provides security engineers, developers and QA engineers the opportunity to develop a mutual understanding of the risks (perceived or real) associated with a given technology, service or system
- Threat modeling makes it easier to find and address vulnerabilities because it requires that the participants systematically evaluate how their components could be attacked and identify the implications of those attacks.

- Procedure

- The process of creating a threat model can be thought of as decomposing an application to understand connectivity and data flow between components that could be exploited by an attacker, it can be boiled down to answering four key questions:
 1. What are we building?
 2. What can go wrong?
 3. What are we going to do about it?
 4. Did we do a good job?

Threat Modeling cont.

- Scoping the Model

- "What are we building" (scoping) is one of the most important steps in threat modeling
- The boundaries of a threat model should be well defined and understood
 - Without a well-defined scope, threat models can become large and unfocused making it difficult to identify and mitigate threats
- To understand the scope of the model, you would need to review process flow diagram(PFD), data flow diagrams(DFD), and architecture diagrams (AD) that you are threat modeling

- Identifying Threats

- To answer "what can go wrong", we need to assume the role of an attacker
- If you knew how the application was designed, how and where would you attack it

- Additional Considerations

- After identifying a threat, consider these questions:
 - If you were an attacker, how would you take advantage of this
 - What is the easiest (from attacker's perspective) attack vector
 - Is this threat already mitigated (partially or completely) by other controls
 - Could this threat affect company and teams reputation
 - Could this threat affect workforce productivity
 - How does this threat affect the confidentiality of your component, or the data it operates on
 - How does this threat affect the integrity of your component or our software stack
 - How does this threat affect the availability of your component or our software stack
 - Has this threat been exploited before

Threat Modeling Case Study

Below is a simplified threat modeling example

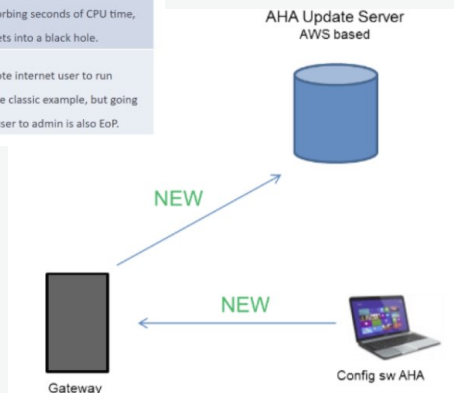
- It starts with documenting data-flow diagram for a web-based system
- Then identifying trusted boundaries and applying STRIDE analysis to it

Use Case

- A company "AHA" is designing an Update Service to securely update their devices remotely
- In the first stage this will be used to update the on-premise gateway
- In later stages other devices will also be updated through this Update Service
- It will use AWS to deploy this new Update Service

STRIDE Overview

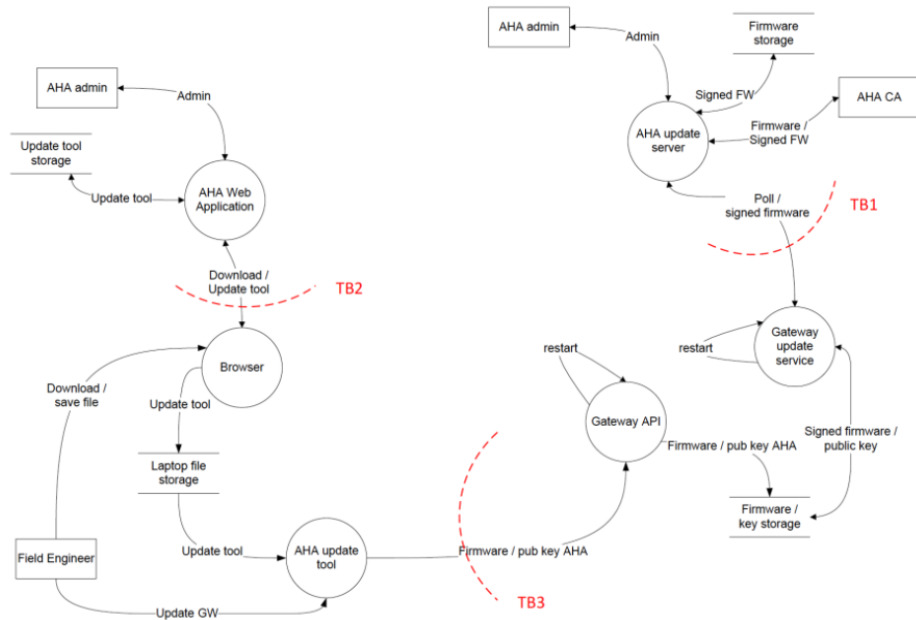
Property	Threat	Definition	Example
Authentication	Spoofing	Impersonating something or someone else.	Pretending to be any of billg, microsoft.com or ntdll.dll
Integrity	Tampering	Modifying data or code	Modifying a DLL on disk or DVD, or a packet as it traverses the LAN.
Non-repudiation	Repudiation	Claiming to have not performed an action.	"I didn't send that email," "I didn't modify that file," "I <i>certainly</i> didn't visit that web site, dear!"
Confidentiality	Information Disclosure	Exposing information to someone not authorized to see it	Allowing someone to read the Windows source code; publishing a list of customers to a web site.
Availability	Denial of Service	Deny or degrade service to users	Crashing Windows or a web site, sending a packet and absorbing seconds of CPU time, or routing packets into a black hole.
Authorization	Elevation of Privilege	Gain capabilities without proper authorization	Allowing a remote internet user to run commands is the classic example, but going from a limited user to admin is also EoP.



Threat Modeling Case Study cont.

Diagram

A 2-in-1 data-flow diagram with the trust boundaries identified and highlighted in red:



STRIDE Analysis

Documenting STRIDE analysis results for each trust boundaries: TB1, TB2 & TB3. Potential threats are highlighted in red:

TB1	GW update service	GWUS-US dataflow	Update server
S	Username / pw (SSL)	x	SSL cert
T	Signed firmware	SSL	Injected firmware = brick all GWs? Remote control all GWs?
R	No logging	x	Audit trail?
I	Leaking of credentials (reverse engineering)	SSL	Leaking of the firmware?
D	Depends on local network	HA Internet link	HA setup server
E	-	x	Separate admin GUI

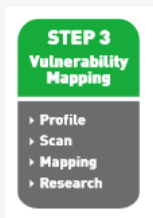
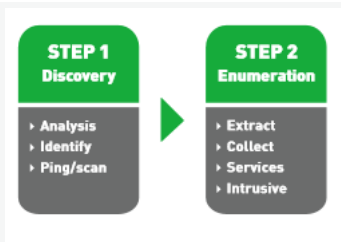
TB2	Browser	Brow-AWA dataflow	Web application
S	Username / password (SSL)	x	SSL cert
T	-	SSL	Tampered Update Tool?
R	No logging	x	Audit trail?
I	-	SSL	Leaking of the firmware?
D	Depends on local network	HA Internet link	HA setup server
E	Unauthorized users getting access as field eng	x	Separate admin GUI

TB3	Update tool	UT-GW API dataflow	GW API
S	Hardcoded user/pw	x	Hardcoded U/P
T	Not signed	Clear text	Rogue firmware possible = brick GW
R	No logging	x	Audit trail?
I	Leaking of hardcoded User/PW	Clear text	Detectable on network (local)
D	Depends on local network	Depends on local network	Depends on local network
E	Unauthorized users getting access as to the API	x	API can be abused to gain access

Penetration Testing

- Penetration testing is a process in which evaluators attempt to circumvent the security features of a system based on their understanding of the system design and implementation
- The purpose is to identify methods of gaining access to a system by using common tools and techniques used by attackers
- Although our testers use well-known testing products such as Nessus, Qualys, Metasploit, AppScan, OWASP ZAP, Burp Suite Pro, we're aware that tools are not always fully capable of mimicking the thought processes and behavior of human testers
- Our testing methodology emphasizes proprietary manual testing techniques and vulnerability mapping, and our unique penetration testing approach is flexible and tailored to meet each assessment's particular needs

Penetration Testing cont.

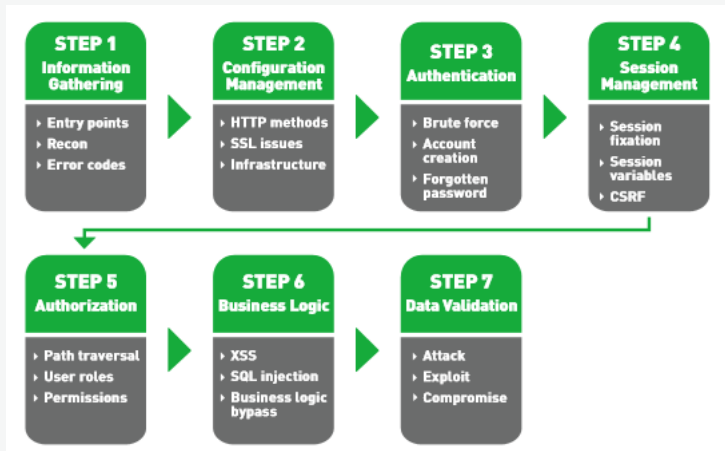


- **Discovery & Enumeration**
 - Various tools and techniques are used to determine whether the network elements are available and actually “listening” (i.e., active or alive)
 - Each of the identified IP addresses is “port-scanned” to identify the active (and potentially exploitable) network services for each element
- **Vulnerability Mapping**
 - During this step, the tester maps the profile of the environment to publicly known, or in some cases, un-documented vulnerabilities
- **Exploitation**
 - The objective of this step is to exploit the vulnerabilities identified in previous steps
 - This is done by using several databases documenting known exploits, including any internally-identified zero-day exploits
 - Trying to obtain unauthorized access to the systems, via un-documented vulnerabilities

Dynamic Analysis Security Testing (DAST)

- DAST involves testing the application from the ***outside in***
 - By examining the application in **its running state** and attempting to manipulate it in unexpected ways in order to discover security vulnerabilities
- This type of testing identifies highly-exploitable vulnerabilities such as
 - SQL injection
 - Cross-site scripting
- It also finds runtime issues that can't easily be found by looking at code in its offline state via static analysis, such as
 - Authentication issues
 - Server misconfiguration issues
 - Vulnerabilities that are only visible when you log in as a known user

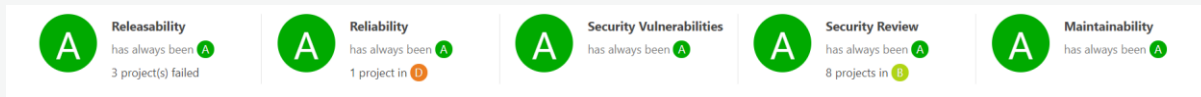
DAST cont.



- Information Gathering
 - During this step, the tester is focused on gaining a clear understanding of how the application functions
- Configuration Management
 - Focused on assessing the infrastructure used in the delivery of the application (e.g. Tomcat, Apache) as well as any database systems used to store application
- Application Testing (e.g. Authentication, Session Management)
 - The goal of these steps is to identify any application-based vulnerabilities that could potentially be exploited
 - Client-side controls
 - Authentication and authorization mechanisms (e.g. roles and permissions)
 - Session management mechanisms
 - Input-based filtering
 - Business logic
 - Use of third-party libraries
 - SQL injection. Path traversal. Cross-site Scripting (XSS). Cross-site Request Forgery (CSRF)

Static Analysis Security Testing (SAST)

- Static application security testing involves testing the application from the **inside out**
 - By examining its source code, byte code or application binaries for conditions indicative of a security vulnerabilities
- Our testers perform manual code reviews and/or using an automated static analysis tools
 - SonarQube (Forcepoint code)



- Artifactory JFroy Xray (3rd Party & Open Source Libraries)

Vulnerabilities Report Example Results

Reports > rbt1 Export

1 out of 5 >

CVE	Summary...	Severity	CVSS:2	CVSS:3	Vulnerabil...	Impacted ...	Path	Fixed Ver...	Published...
N/A	Pivotal Sp...	High	7.5		org.spring...	xray-testv2	release-bu...	3 5.1.5...	08-04-19 1...
CVE-201...	Plexus-util...	High	7.5	9.8	org.codehaus...	xray-testv2	release-bu...	1 3.0.16	10-06-19 1...

- This testing process also involves the manual review of un-compiled source code
 - During this process, the tester will focus on identifying, tracking and recommending solutions to resolve technical and logical security issues

Fuzzing (Fuzz Testing)

- Introduction

- A technique where randomized inputs are provided to a piece of software with the intent of discovering unknown defects
- It's part of Dynamic Analysis Security Testing (remember outside in)
- Designed to uncover defects or (vulnerabilities) which are difficult to identify from
 - Human analysis (code reviews)
 - Static Analysis tools
- Fuzzing efficacy improves by increasing code-coverage

- Requirements

- Clearly defined “target” i.e., whole application, selected functionality
- Target must come with clear documentation on input data it normally expects to receive
- Tools will vary but there are many good tools to choose from depending on what is being tested
- Monitoring is required to properly identify test cases that caused system crash or fault
- Code coverage requires a large and high-quality input set

Vulnerabilities and CVSS

- All vulnerabilities are “scored” using
 - [Common Vulnerability Scoring System Version 3.1 Calculator](#)
 - CVSS is an open framework for communicating characteristics and severity of software vulnerabilities
 - CVSS consists of three metric groups
 - Base – exploitable and impact metrics
 - Temporal – exploit code maturity, remediation level, report confidence
 - Environmental – modified base metrics, confidentiality, integrity and availability requirements
 - Each metric has an impact on the overall score and can either raise or lower it accordingly
 - We work closely with the product teams to score the vulnerability and look for input on
 - Temporal
 - Environmental
 - Each vulnerability has an SLA to fix based on the score
 - Vulnerability metric dashboards are used to track, measure and provide potential technical risk impact

CVSS 3.1 Rating	CVSS 3.1 Score
Critical	9.0 -10.0
High	7.0 – 8.9
Medium	4.0 – 6.9
Low	0.1 – 3.9

References for additional information

Application Security Podcasts – [Application Security Journey](#)

Common Vulnerabilities and Exposures (CVE) – <https://cve.mitre.org/>

Common Vulnerability Scoring System Version 3.1 Calculator (CVSS) – <https://www.first.org/cvss/calculator/3.1>

Common Weakness Enumeration (CWE) - <https://cwe.mitre.org/about/index.html>

Forum of Incident Response & Security Teams (FIRST) – <https://first.org/about>


Hackable Podcasts - <https://hackablepodcast.com/episodes>

Microsoft Threat Modeling - <https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling>

National Institute of Standards and Technology (NIST) – <https://csrc.nist.gov/Topics/Security-and-Privacy>

Open Web Application Security Project (OWASP) – <https://owasp.org>

Security Now Podcasts – <https://twit.tv/shows/security-now>



Thank You

