Ciclo di vita per lo sviluppo di software sicuro

(a cura di **Alessandro Garsia** – **Postecom Spa**)

INDICE DELLA PRESENTAZIONE :

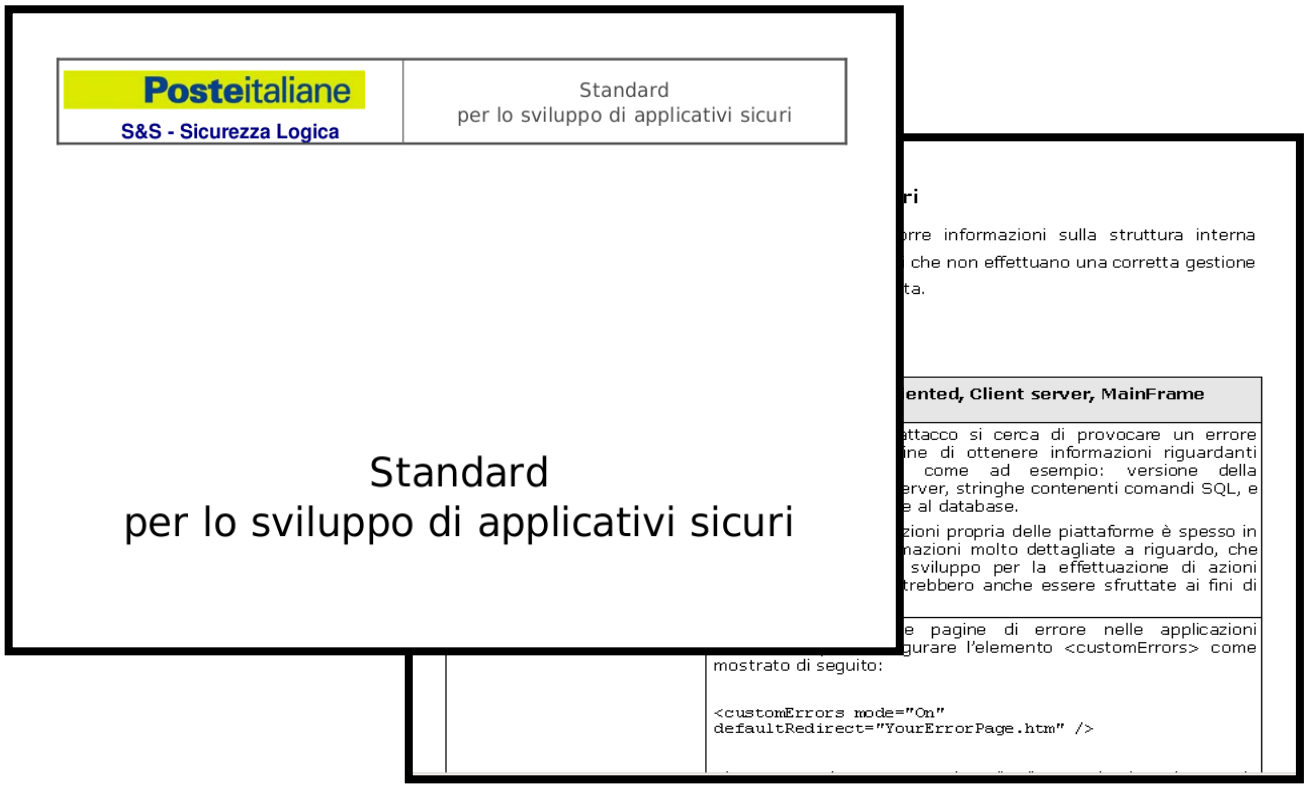
1. Sviluppo di software sicuro: introduzione
2. L'analisi statica del codice sorgente
3. L'analisi dinamica in fase di test
4. L'analisi dinamica delle applicazioni in produzione

Le motivazioni del cambiamento 1/3

Aumentare il livello di sicurezza del software prodotto, riducendo la possibilità di danni di immagine / economici

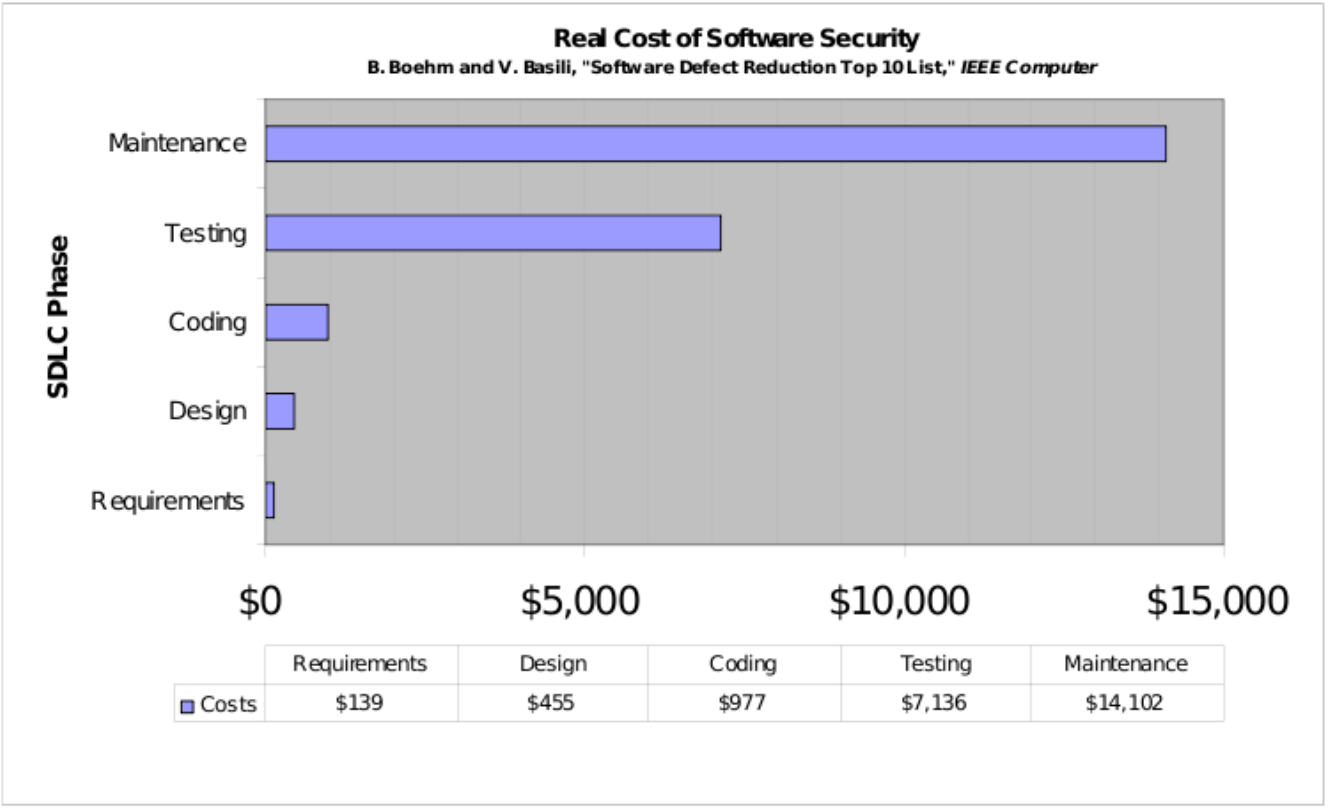
Le motivazioni del cambiamento 2/3

Facilitare l'aderenza allo Standard aziendale per lo sviluppo di applicativi sicuri

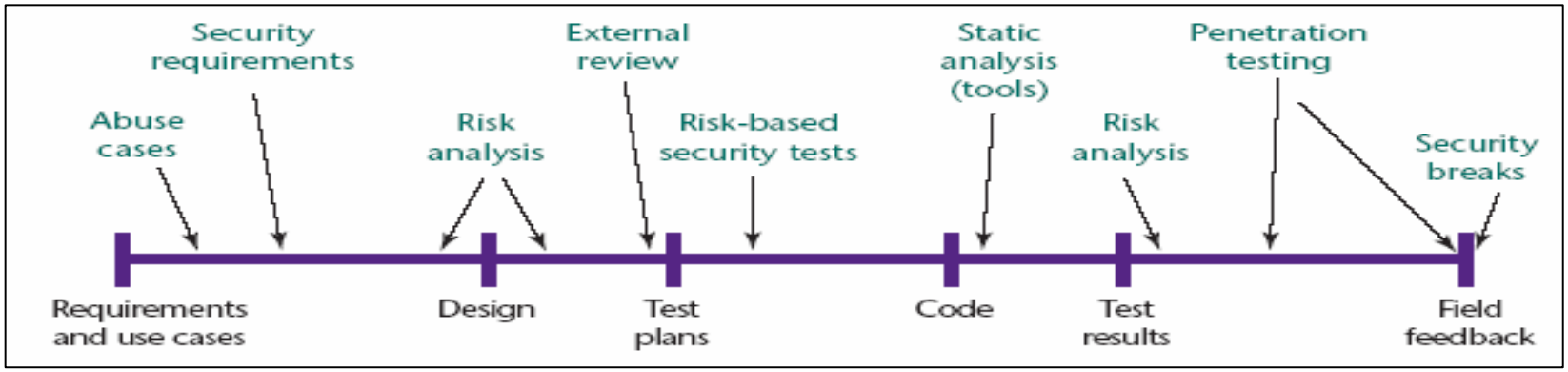


Le motivazioni del cambiamento 3/3

Ridurre l'effort ed i costi necessari per la correzione delle vulnerabilità



Processo di vita del software (fonte IEEE)

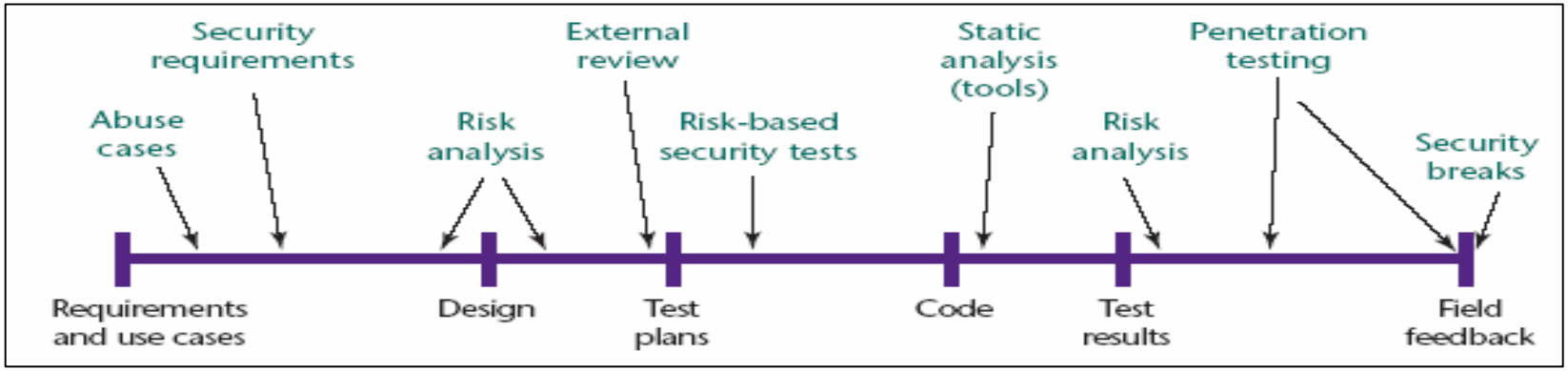


La sicurezza nel processo di Software Development Life Cycle (SDLC) non può essere pensata come una semplice aggiunta alla fine del ciclo di vita, magari già in erogazione.

Lungo tutto il ciclo di vita del software occorre considerare differenti azioni specificamente pensate per indirizzare i problemi di sicurezza.

Il processo di sviluppo sicuro del codice deve seguire procedure organizzative dedicate, pur integrandosi con quelle esistenti, si deve adattare agli strumenti di configurazione e sviluppo esistenti, deve essere supportato da appositi strumenti tecnologici.

Il processo “sicuro” si adatta al contesto



Il S(ecure)DLC richiede una introduzione graduale e basata su obiettivi realistici.

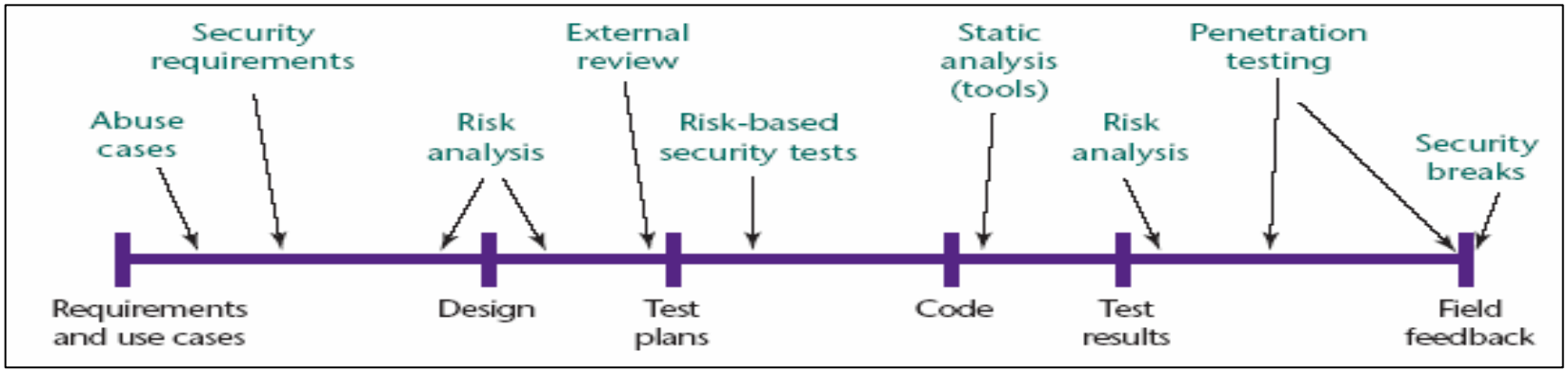
Si può intervenire su un “parco software” esistente oppure su applicazioni da sviluppare ex-novo.

Il software può essere sviluppato da personale interno o può essere commissionato all’esterno.

A seconda dei casi, alcune delle azioni suggerite di seguito potrebbero non essere applicabili.

In primo luogo dovrà essere definito, a seguito di un SDLC Assessment, un piano di intervento adeguato all’ambito in analisi.

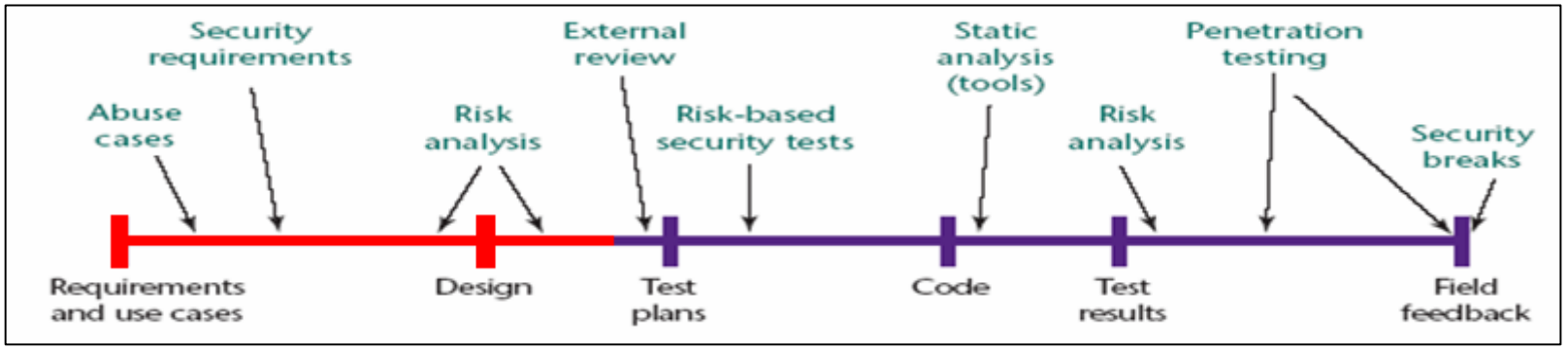
Piano di intervento: esempio individuazione attività



Lo scenario completo di azione è l'intervento sulle applicazioni sviluppate internamente:

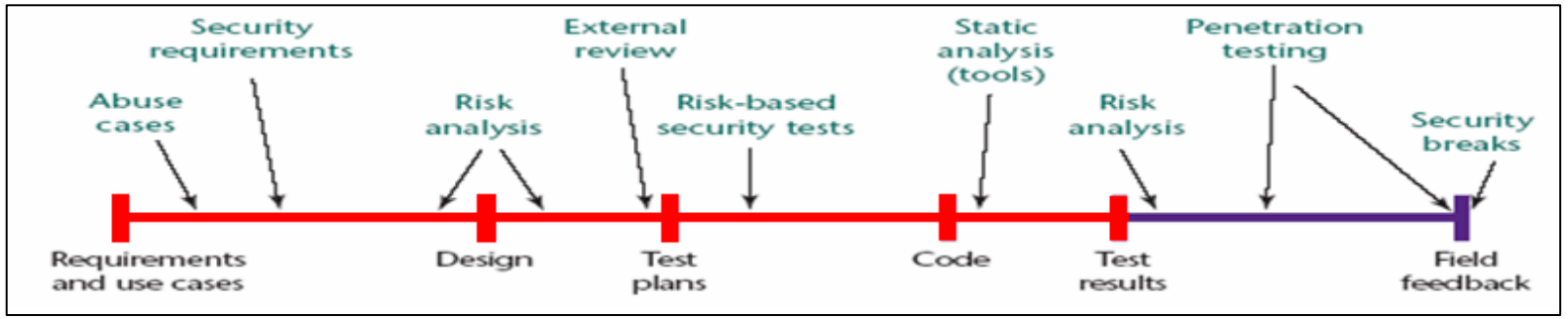
Abuse cases	Security requirements	Risk analysis	External review	Risk-based security tests	Static analysis	Penetration testing	Security breaks
X	X	X	X	X	X	X	X

Fasi "Requirements and use cases" e "Design"



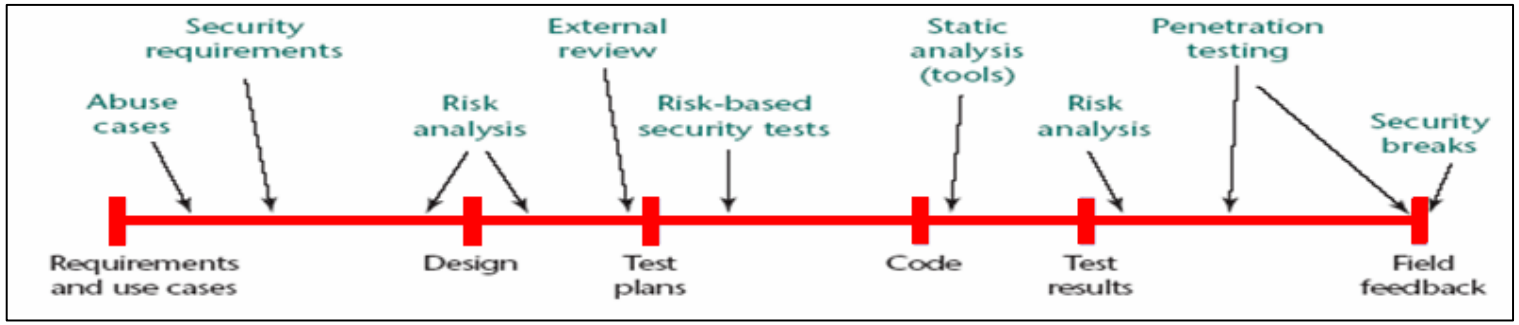
- ***Abuse cases**:* definizione delle "Linee Guida alla programmazione sicura", dettagliate anche per linguaggio di programmazione e per ambiti di utilizzo, da fornire agli sviluppatori interni ed ai fornitori esterni.
- ***Risk analysis**:* definizione di una metodologia coerente con il ciclo di vita del software e che tenga conto della classificazione dei dati.
- ***Security requirements**:* definizione dei requisiti di sicurezza, in base ai risultati della risk analysis e alla classificazione dei dati. Possono contenere, nel caso di fornitori esterni, requisiti legati all'esperienza, alla metodologie, alle competenze del fornitore.

Fase “Code”



- ***Static analysis****:* identificazione dei tool di analisi statica in grado di:
  - controllare il codice sorgente in base a delle regole built-in, ovviamente aggiornabili, e configurabili secondo la realtà del SDLC in cui sono inseriti;
  - lavorare su più linguaggi di programmazione e su più moduli dell'applicazione contemporaneamente;
  - creare una mappa dei componenti dell’architettura di un software per controllare i collegamenti e il flusso di controllo, consentendo di capire dove impattano dei dati non validati;
  - creare dei report destinati ai diversi soggetti coinvolti (sviluppatori, sw architect, security manager, etc…);
  - essere integrati nell’ambiente di sviluppo (plug-in per IDE) utilizzato dai programmatori in modo da segnalare la presenza di porzioni di codice non corrette, dal punto di vista della sicurezza, nello stesso istante in cui il codice sorgente stesso viene scritto.

Fasi "Test plans", "Test results" e "Field feedback"



- ***Risk-based security tests**:*
  - definizione delle metriche – misurazione di una serie di parametri che vengono poi pesati per produrre degli indici che sintetizzino la situazione dei propri progetti e team di sviluppo;
  - identificazione dei tool di Penetration testing integrabili con gli strumenti di Q&A esistenti.

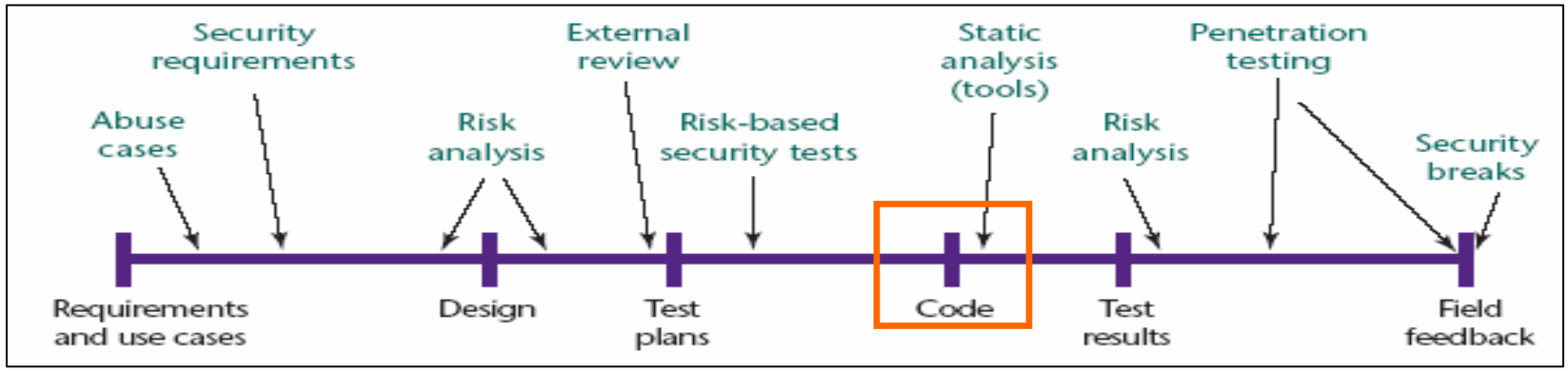
- ***Penetration testing**:* esecuzione di analisi di vulnerabilità.

- ***Security Breaks**:* valutazione (e risoluzione – Gestione degli Incidenti) delle eventuali vulnerabilità segnalate in fase di erogazione, così da effettuare il tuning degli abuse case e degli strumenti impiegati.

INDICE DELLA PRESENTAZIONE :

1. Sviluppo di software sicuro: introduzione
2. L'analisi statica del codice sorgente
3. L'analisi dinamica in fase di test
4. L'analisi dinamica delle applicazioni in produzione

L'analisi statica del codice sorgente



Analisi statica: vantaggi

***Static code analysis advantages**:*

- It permits weaknesses to be found earlier in the development life cycle, reducing the cost to fix.
- It can find weaknesses in the code at the exact location.
- It can be conducted by trained software assurance developers who fully understand the code.
- It allows a quicker turn around for fixes.
- It is relatively fast if automated tools are used.
- Automated tools can scan the entire code base.
- Automated tools can provide mitigation recommendations, reducing the research time.

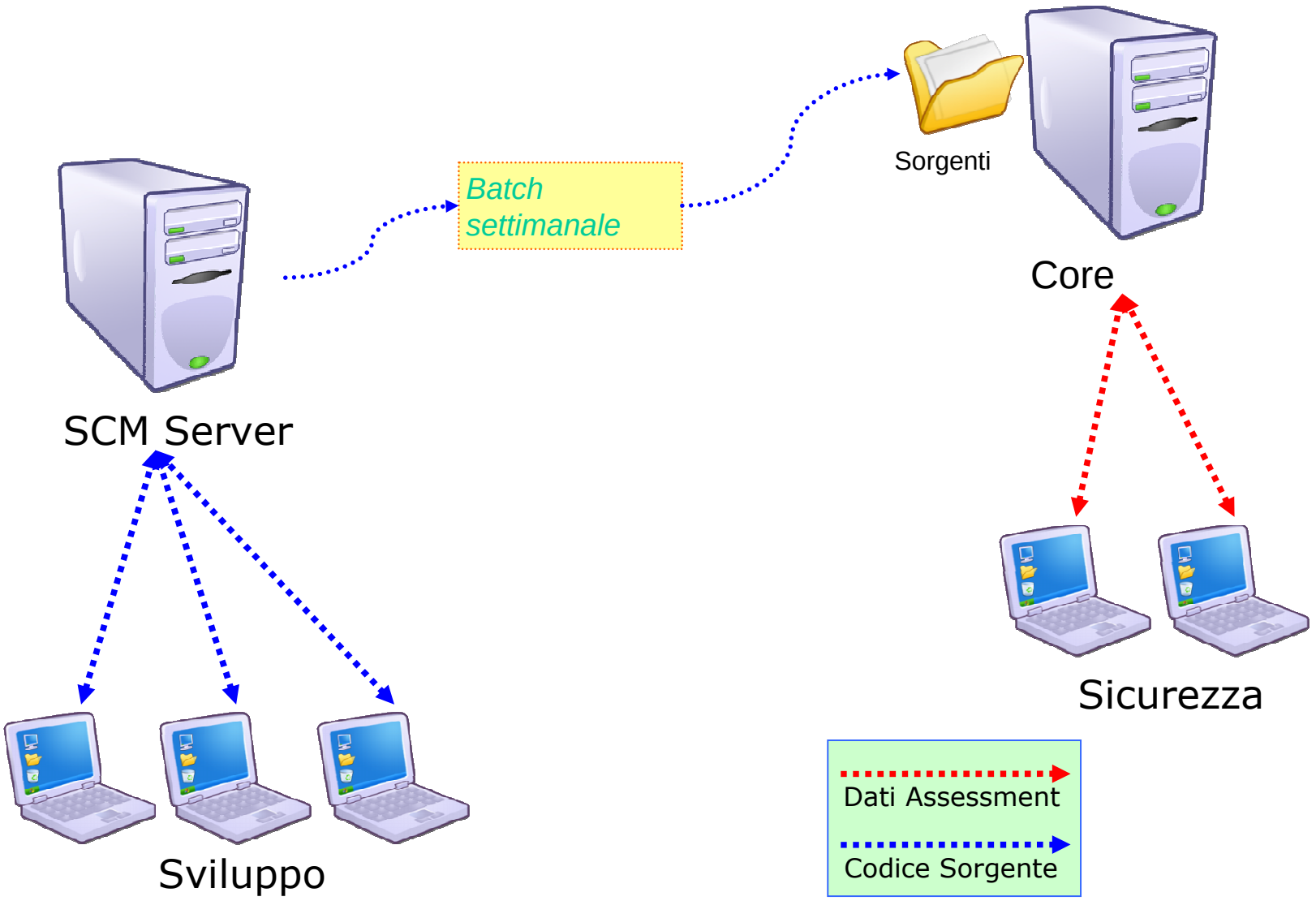
Fonte: Government Computer News

***Static code analysis limits**:*

- It is time consuming if conducted manually.
- Automated tools do not support all programming languages.
- Automated tools produce false positives and false negatives.
- There are not enough trained personnel to thoroughly conduct static code analysis.
- Automated tools can provide a false sense of security that everything is being addressed.
- Automated tools only as good as the rules they are using to scan with.
- It does not find vulnerabilities introduced in the runtime environment.

Fonte: Government Computer News

Analisi statica: architettura



Impatto a livello sistemistico

- Basso per i progetti.Net:
  - Gli scan avvengono sul Core:
    - Tutti gli ambienti di build sono riprodotti sul Core

- Basso per i progetti Java:
  - Gli scan avvengono sul Core:
    - Il progetto deve essere compatibile con Maven

Il triage

È l'analisi dei risultati prodotti dall'assessment e messa in opera delle azioni necessarie al fine di eliminare tutte le anomalie registrate.

Si sintetizza in pochi passi:

- Accurata indagine per l'effettivo riscontro delle anomalie;
- Confronto con i referenti tecnici di riferimento;
- Raggruppamento delle anomalie confermate e non;
- Congiuntamente, individuazione della contromisura e assegnazione al programmatore per la risoluzione;
- Esclusione dei "falsi positivi" (quelle anomalie che dopo un'accurata analisi non sono state confermate come vulnerabilità) dalle metriche degli assessment.

I Security Analyst

- **Cosa fanno**
  - Periodicamente, gestiscono i risultati degli assessment dei singoli progetti ("Triage").
  - Possono effettuare "assessment" di tipo provvisorio per verificare l'effettivo buon fine del triage

- **Cosa NON fanno**
  - Non possono autonomamente modificare le policy di sicurezza: severity delle anomalie segnalate, classificazioni, ecc;
  - Non possono pianificare "assessment" utili ai fini delle metriche di sicurezza.

Gli Sviluppatori

- **Cosa fanno (plug-in per IDE):**
  - Importano le liste di Bug/Exceptions a loro assegnate e applicano le correzioni indicate dal Security Analyst sul progetto di riferimento seguendo il normale processo SDLC (CM,Test,Deploy,ecc).
  - Possono effettuare “assessment” di tipo provvisorio per verificare l’effettivo buon fine delle correzioni.
  - Eventualmente supportano il Security Analyst nell’analisi dei risultati.

- **Cosa NON fanno**
  - Non effettuano autonomamente l’analisi dei risultati dell’assessment.
  - Non possono consolidare “assessment” utili per le metriche di sicurezza.

Impatto sui processi

- Realizzazione di scan automatici con cadenza settimanale:
  - Es.: il venerdì sera:
- Assessment a disposizione dei Security Analys:
  - Es.: Il lunedì mattina
- Triage
- Al successivo scan su un certo progetto
  - Se esiste un assessment “recente” lo si pubblica;
  - altrimenti si pubblica l’assessment realizzato con lo scan.

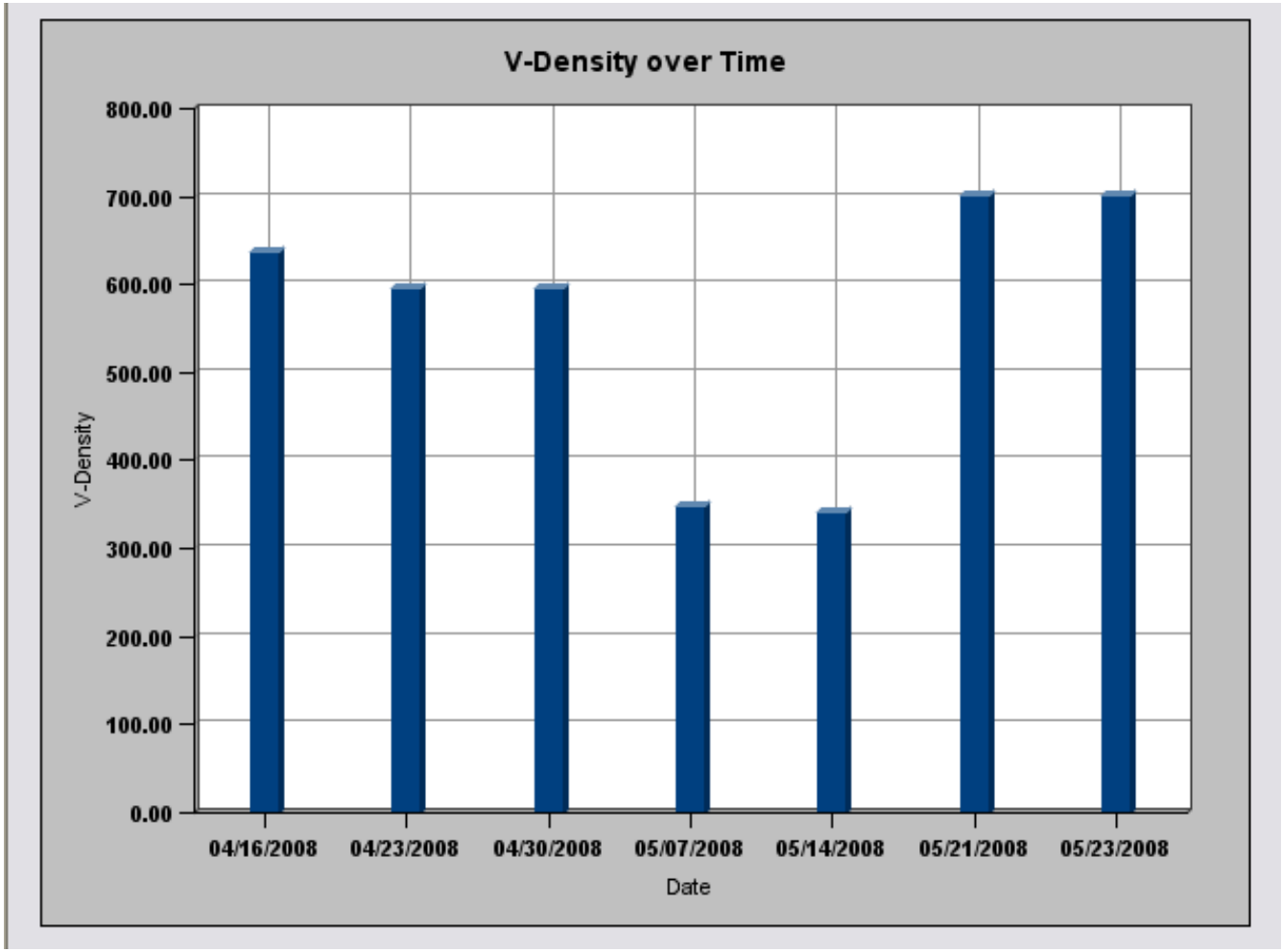
Analisi statica: la situazione iniziale

Top Vulnerable Applications

	Application	Assessment		Vulnerabilities			Type I			Type II		
				High	Medium	Low	High	Medium	Low	High	Medium	Low
		5-mag-2008	3.461,18		3		694	61	30	9	4	114
		5-mag-2008	3.202,08		3		690	61	30	9	4	114
		13-mag-2008	1.033,63	5	6	31	1	26	43	9	92	86
		16-mag-2008	700,44	1			107	469	23	1.084	49	55
		6-mag-2008	51,13	3		4	1	1	45	5	177	52
		16-mag-2008	15,02							24		12

All Applications

Analisi statica: l'andamento per l'applicazione



Analisi statica: il primo assessment

		Assessment		Vulnerabilities			Type I			Type II		
				High	Medium	Low	High	Medium	Low	High	Medium	Low
		14-apr-2008	741,47				3	33		70		2
		14-apr-2008	330,15				2			17		1
		14-apr-2008	554,67				4	16		42		1
		14-apr-2008	474,80				6	18		9		3
		14-apr-2008	194,26					6		8	9	
		14-apr-2008	574,95				2	10		23		1
		14-apr-2008	350,74				4	19		5		1
		14-apr-2008	2.423,63	1			8	19		39		
		14-apr-2008	489,91				3	7		13		2
		14-apr-2008	1.092,00				5	24		27		1
		14-apr-2008	660,70				3	132		184		27
		14-apr-2008	288,94				3	15		35	5	
		14-apr-2008	0,00									
		14-apr-2008	949,02				7	13		38		

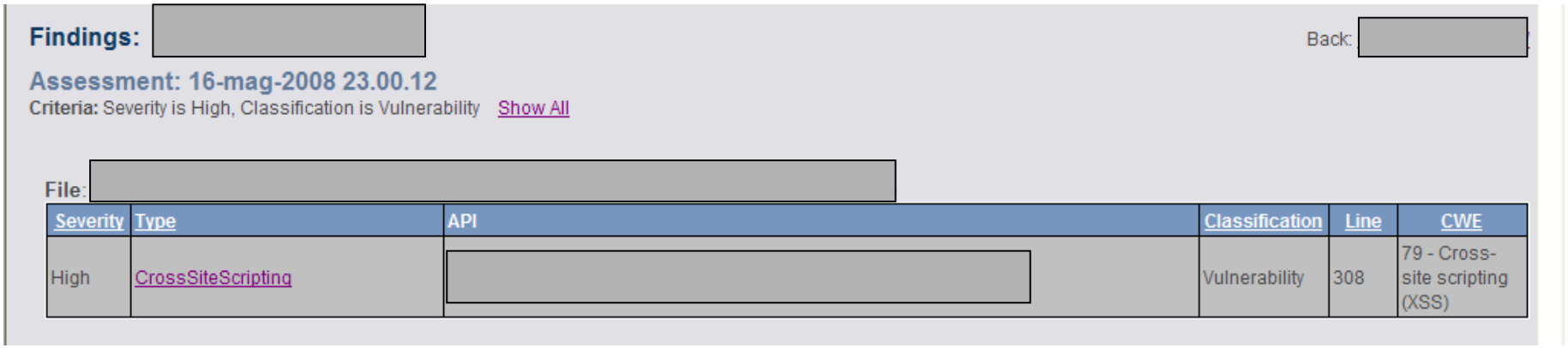
Analisi statica: assessment senza vulnerabilità

		Assessment		Vulnerabilities			Type I			Type II		
				High	Medium	Low	High	Medium	Low	High	Medium	Low
		9-mag-2008	148,61					18		9	2	1
		9-mag-2008	130,38					2		8		
		9-mag-2008	314,64					8		21		1
		9-mag-2008	93,62					13		4		1
		9-mag-2008	587,02					19		38		
		9-mag-2008	194,80					7		12	1	1
		9-mag-2008	485,54					18		27		1
		9-mag-2008	570,04					119		179	1	26
		9-mag-2008	164,51					15		32		
		9-mag-2008	0,00									
		9-mag-2008	395,57					13		37		
		9-mag-2008	565,81					32		69	1	1
		9-mag-2008	53,43							6		
		9-mag-2008	318,75					16		40	1	

Analisi statica: assessment successivo

		Assessment		Vulnerabilities			Type I			Type II		
				High	Medium	Low	High	Medium	Low	High	Medium	Low
		16-mag-2008	587,02					19		38		
		16-mag-2008	485,54					18		27		1
		16-mag-2008	378,18				5	12		61	7	8
		16-mag-2008	1.223,88				10	16		48		
		16-mag-2008	0,00									
		16-mag-2008	1.937,50				10	34	4	64		
		16-mag-2008	1.462,75				7	5	2	54	4	
		16-mag-2008	53,43							6		
		16-mag-2008	1.074,30				9	23		73	16	8
		16-mag-2008	597,39				5	1		4		2
		16-mag-2008	148,61					18		9	2	1
		16-mag-2008	130,38					2		8		
		16-mag-2008	314,64					8		21		1
		16-mag-2008	88,11					12		4		1
		16-mag-2008	2.699,66	1			8	14		46		
		16-mag-2008	414,84					10		20	7	4
		16-mag-2008	2.095,43				15	12	4	44		
		16-mag-2008	194,80					7		12	1	1
		16-mag-2008	570,04					119		179	1	26
		16-mag-2008	914,32				3	18	3	20		
		16-mag-2008	164,51					15		32		
		16-mag-2008	395,57					13		37		
		16-mag-2008	804,11				4	10		44		
		16-mag-2008	154,28					2		9	9	1
		16-mag-2008	565,81					32		69	1	1
		16-mag-2008	318,39					16		40	1	
		16-mag-2008	2.258,85				16	11	4	52		
		16-mag-2008	2.064,86				15	22	6	63		

Analisi statica: la vulnerabilità *high*



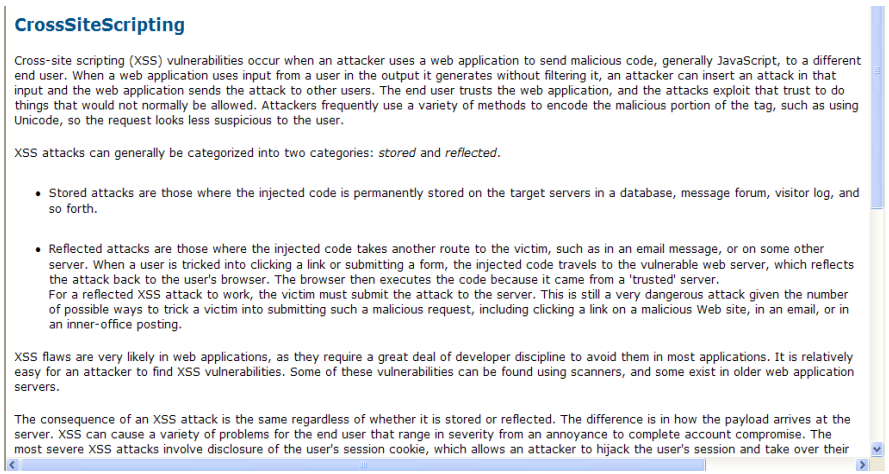
Findings: [redacted] Back: [redacted]

Assessment: 16-mag-2008 23.00.12

Criteria: Severity is High, Classification is Vulnerability Show All

File: [redacted]

Severity	Type	API	Classification	Line	CWE
High	CrossSiteScripting	[redacted]	Vulnerability	308	79 - Cross-site scripting (XSS)



CrossSiteScripting

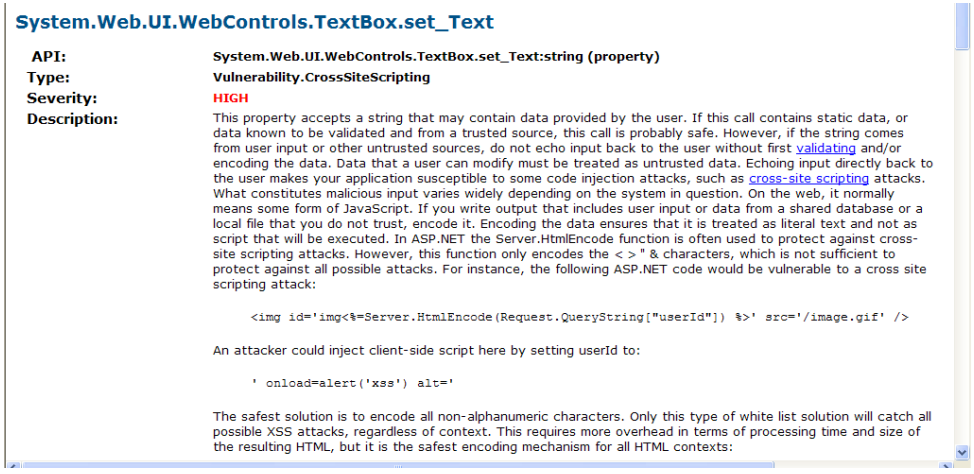
Cross-site scripting (XSS) vulnerabilities occur when an attacker uses a web application to send malicious code, generally JavaScript, to a different end user. When a web application uses input from a user in the output it generates without filtering it, an attacker can insert an attack in that input and the web application sends the attack to other users. The end user trusts the web application, and the attacks exploit that trust to do things that would not normally be allowed. Attackers frequently use a variety of methods to encode the malicious portion of the tag, such as using Unicode, so the request looks less suspicious to the user.

XSS attacks can generally be categorized into two categories: *stored* and *reflected*.

- Stored attacks are those where the injected code is permanently stored on the target servers in a database, message forum, visitor log, and so forth.
- Reflected attacks are those where the injected code takes another route to the victim, such as in an email message, or on some other server. When a user is tricked into clicking a link or submitting a form, the injected code travels to the vulnerable web server, which reflects the attack back to the user's browser. The browser then executes the code because it came from a 'trusted' server.
  For a reflected XSS attack to work, the victim must submit the attack to the server. This is still a very dangerous attack given the number of possible ways to trick a victim into submitting such a malicious request, including clicking a link on a malicious Web site, in an email, or in an inner-office posting.

XSS flaws are very likely in web applications, as they require a great deal of developer discipline to avoid them in most applications. It is relatively easy for an attacker to find XSS vulnerabilities. Some of these vulnerabilities can be found using scanners, and some exist in older web application servers.

The consequence of an XSS attack is the same regardless of whether it is stored or reflected. The difference is in how the payload arrives at the server. XSS can cause a variety of problems for the end user that range in severity from an annoyance to complete account compromise. The most severe XSS attacks involve disclosure of the user's session cookie, which allows an attacker to hijack the user's session and take over their



System.Web.UI.WebControls.TextBox.set_Text

API: System.Web.UI.WebControls.TextBox.set_Text:string (property)
Type: Vulnerability.CrossSiteScripting
Severity: HIGH
Description: This property accepts a string that may contain data provided by the user. If this call contains static data, or data known to be validated and from a trusted source, this call is probably safe. However, if the string comes from user input or other untrusted sources, do not echo input back to the user without first validating and/or encoding the data. Data that a user can modify must be treated as untrusted data. Echoing input directly back to the user makes your application susceptible to some code injection attacks, such as cross-site scripting attacks. What constitutes malicious input varies widely depending on the system in question. On the web, it normally means some form of JavaScript. If you write output that includes user input or data from a shared database or a local file that you do not trust, encode it. Encoding the data ensures that it is treated as literal text and not as script that will be executed. In ASP.NET the Server.HtmlEncode function is often used to protect against cross-site scripting attacks. However, this function only encodes the < > " & characters, which is not sufficient to protect against all possible attacks. For instance, the following ASP.NET code would be vulnerable to a cross site scripting attack:

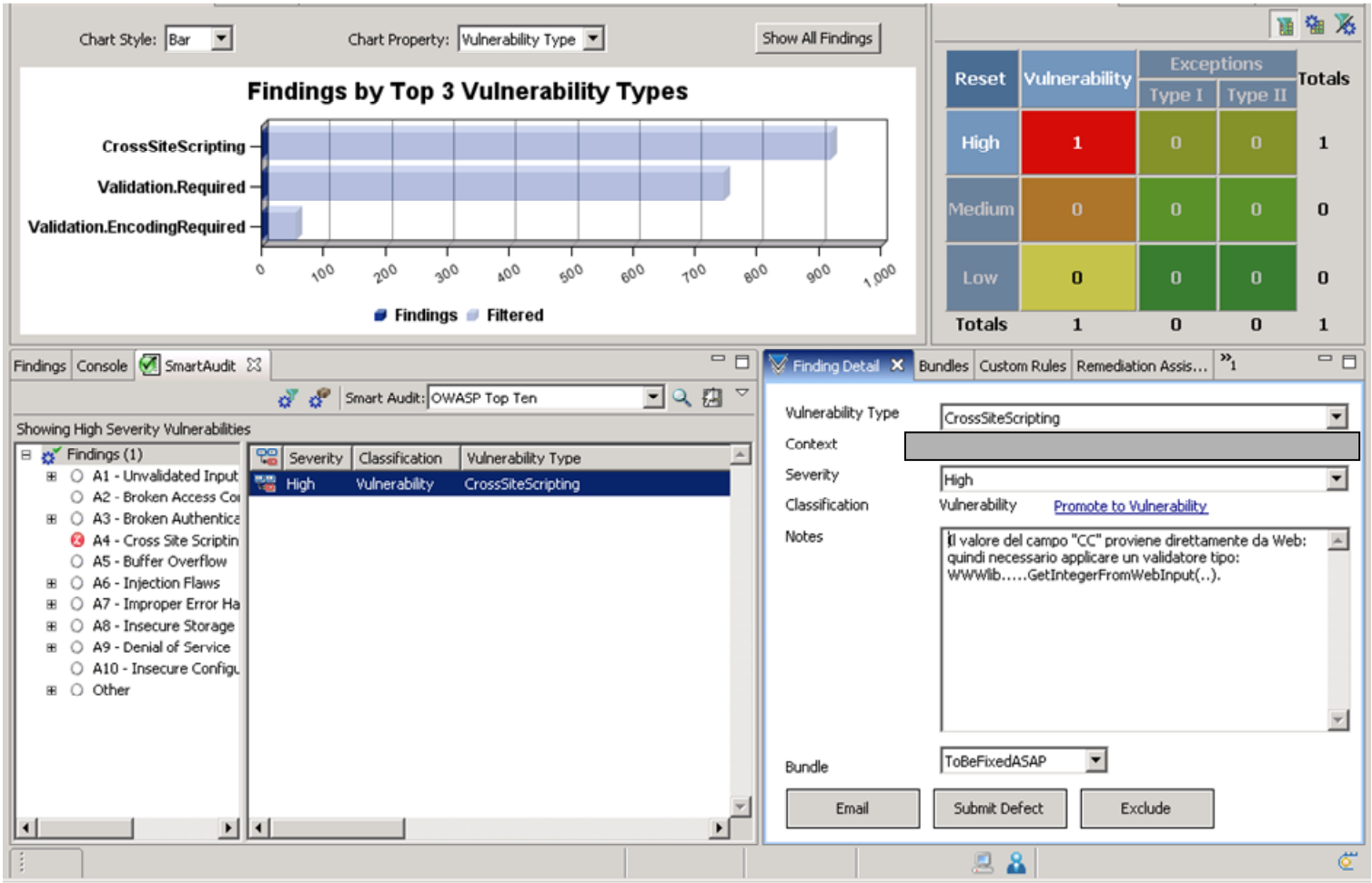
```
<img id='img<%=Server.HtmlEncode(Request.QueryString["userId"]) %>' src='/image.gif' />
```

An attacker could inject client-side script here by setting userId to:

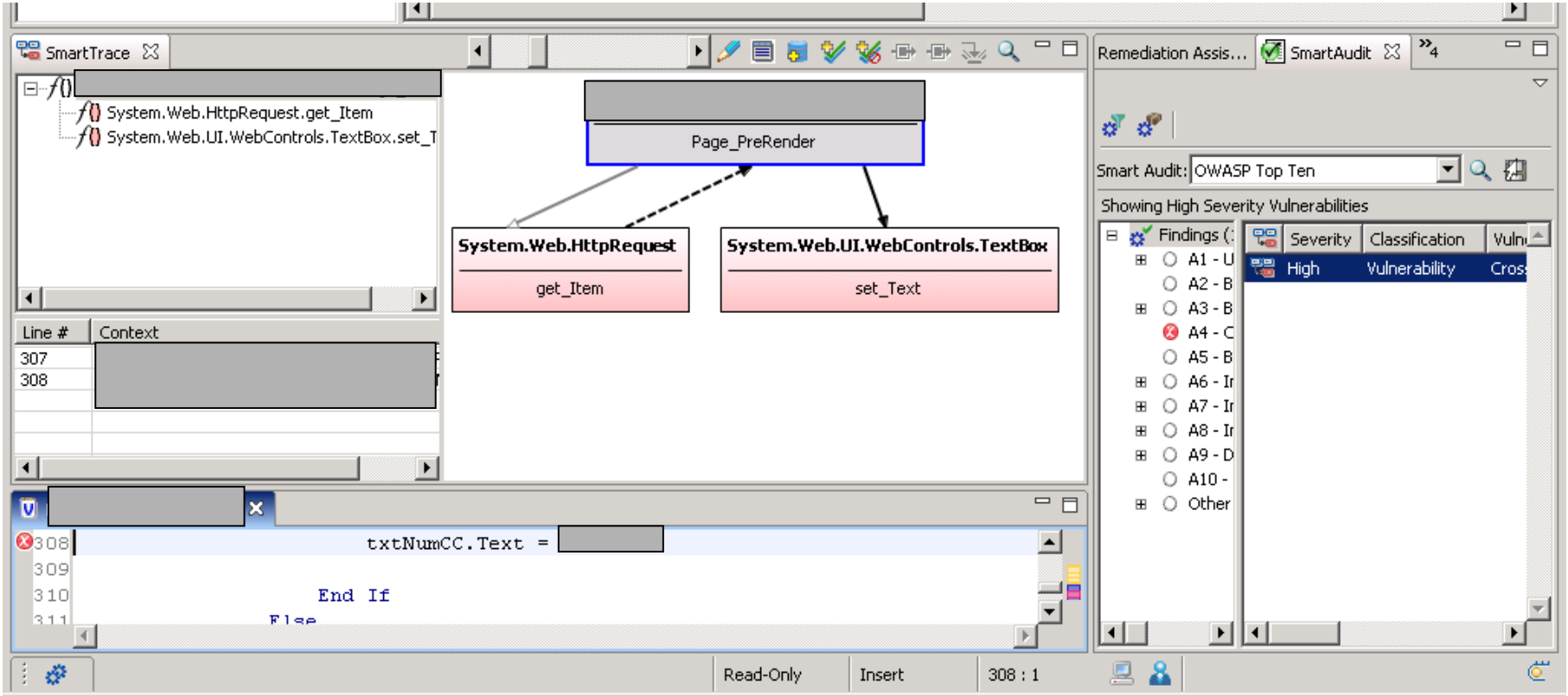
```
' onload=alert('xss') alt='
```

The safest solution is to encode all non-alphanumeric characters. Only this type of white list solution will catch all possible XSS attacks, regardless of context. This requires more overhead in terms of processing time and size of the resulting HTML, but it is the safest encoding mechanism for all HTML contexts:

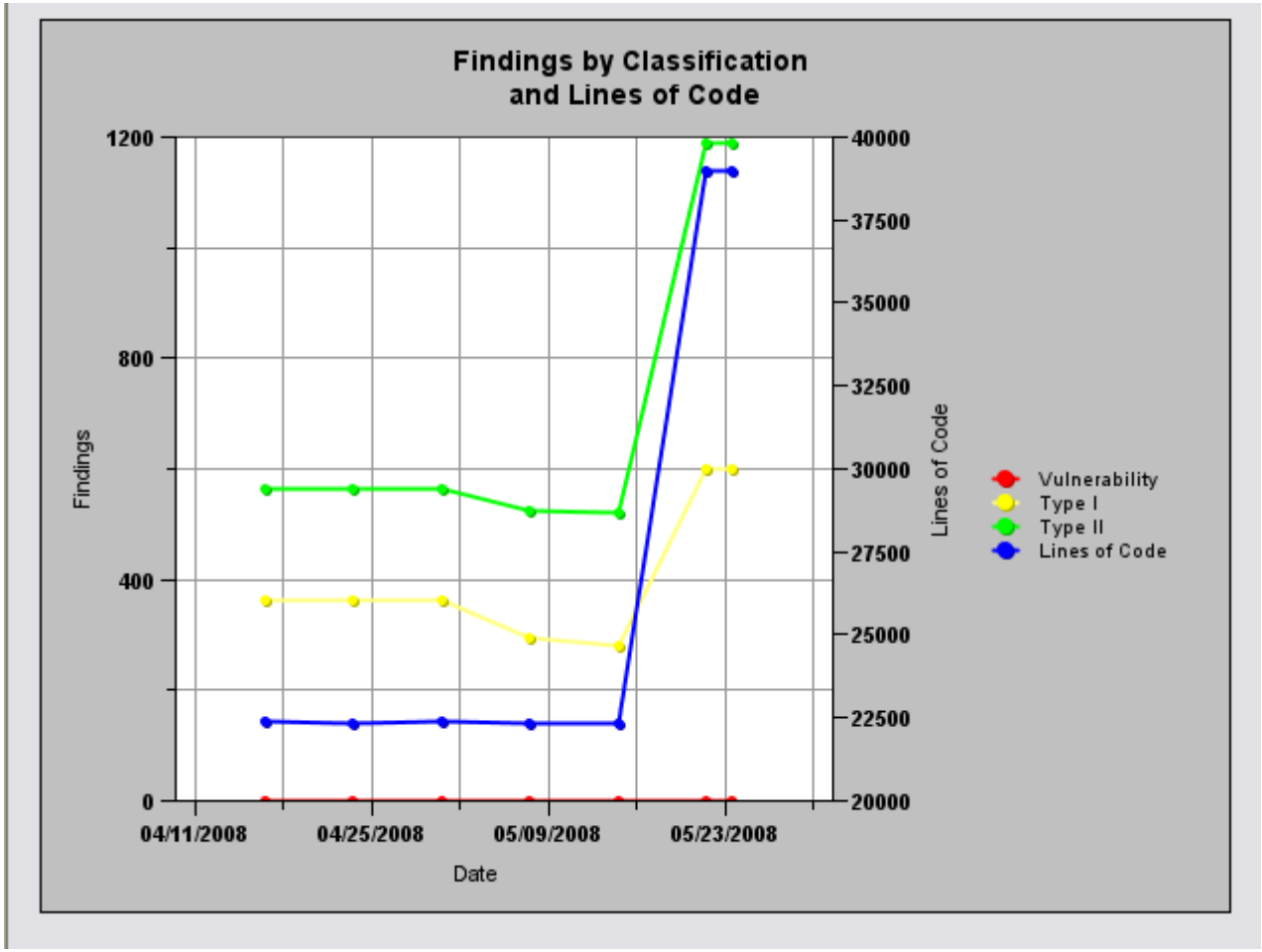
Analisi statica: la console 1/2



Analisi statica: la console 2/2



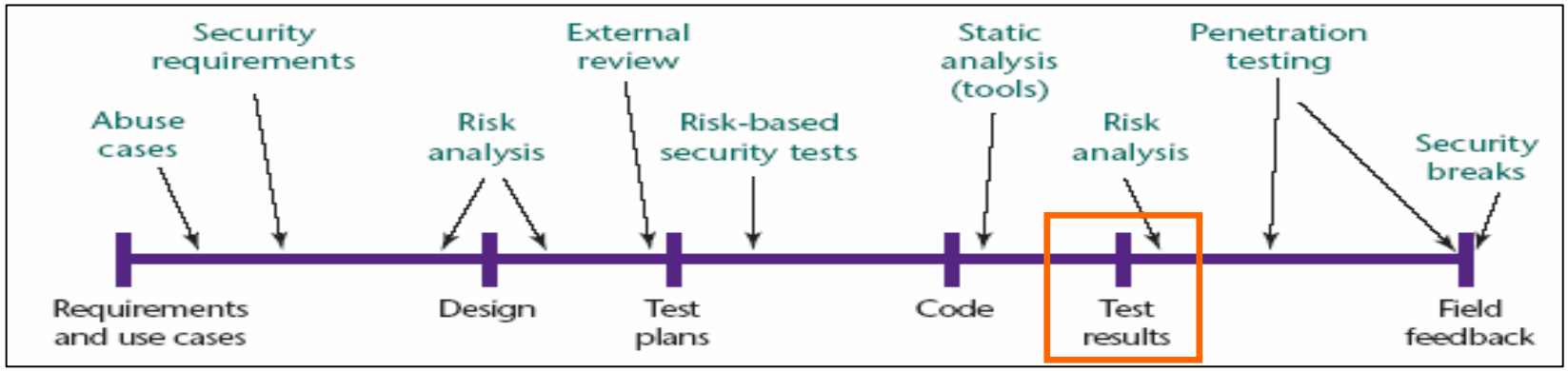
Analisi statica: “errori” per KLOC



INDICE DELLA PRESENTAZIONE :

1. Sviluppo di software sicuro: introduzione
2. L'analisi statica del codice sorgente
3. L'analisi dinamica in fase di test
4. L'analisi dinamica delle applicazioni in produzione

L'analisi dinamica in fase di test



Analisi dinamica in fase di test: vantaggi

Dynamic code analysis advantages:

- It identifies vulnerabilities in a runtime environment.
- Automated tools provide flexibility on what to scan for.
- It allows for analysis of applications in which you do not have access to the actual code.
- It identifies vulnerabilities that might have been false negatives in the static code analysis.
- It permits you to validate static code analysis findings.
- It can be conducted against any application.

Fonte: Government Computer News

Dynamic code analysis limits:

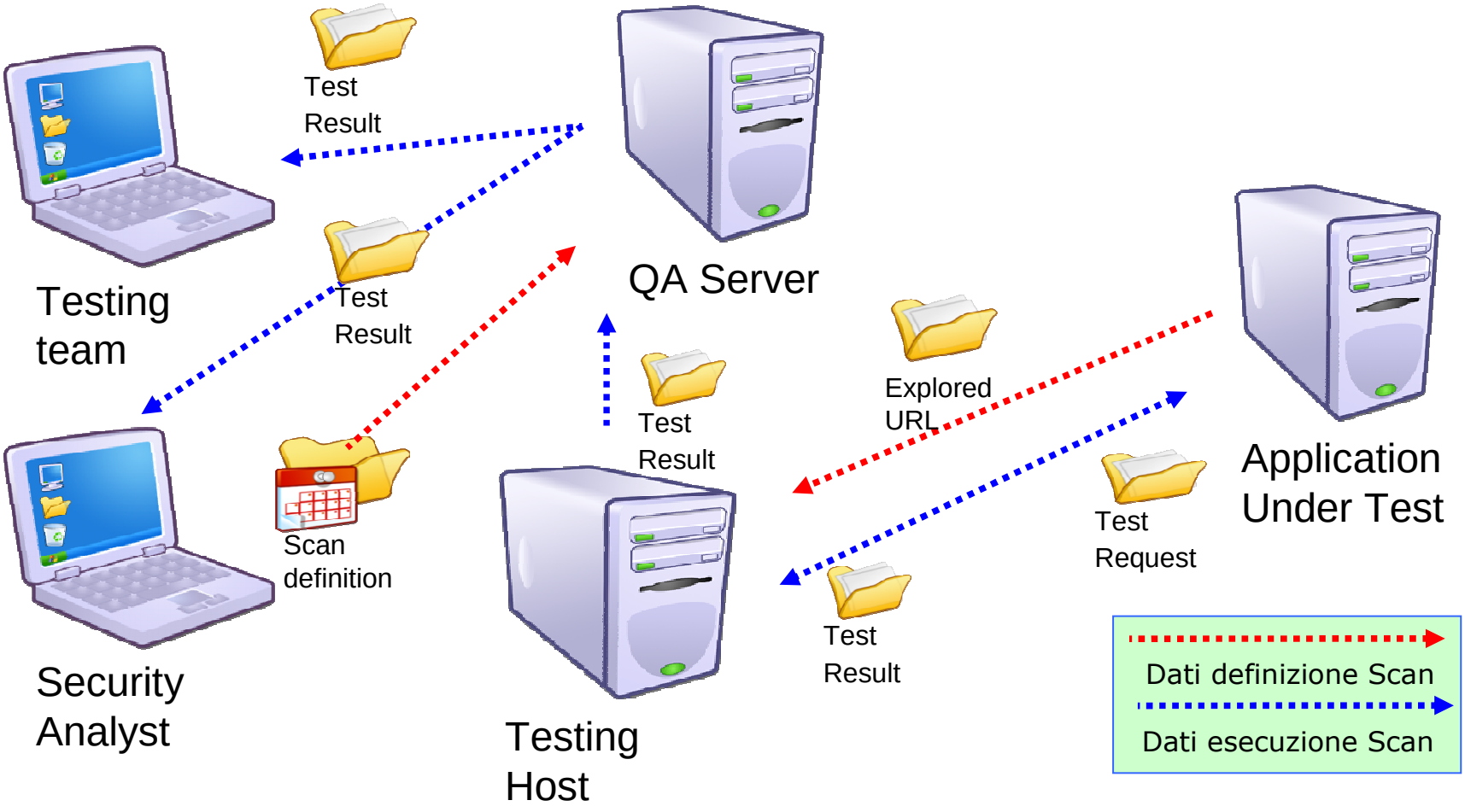
- Automated tools provide a false sense of security that everything is being addressed.
- Automated tools produce false positives and false negatives.
- Automated tools are only as good as the rules they are using to scan with.
- There are not enough trained personnel to thoroughly conduct dynamic code analysis [as with static analysis].
- It is more difficult to trace the vulnerability back to the exact location in the code, taking longer to fix the problem.

Fonte: Government Computer News

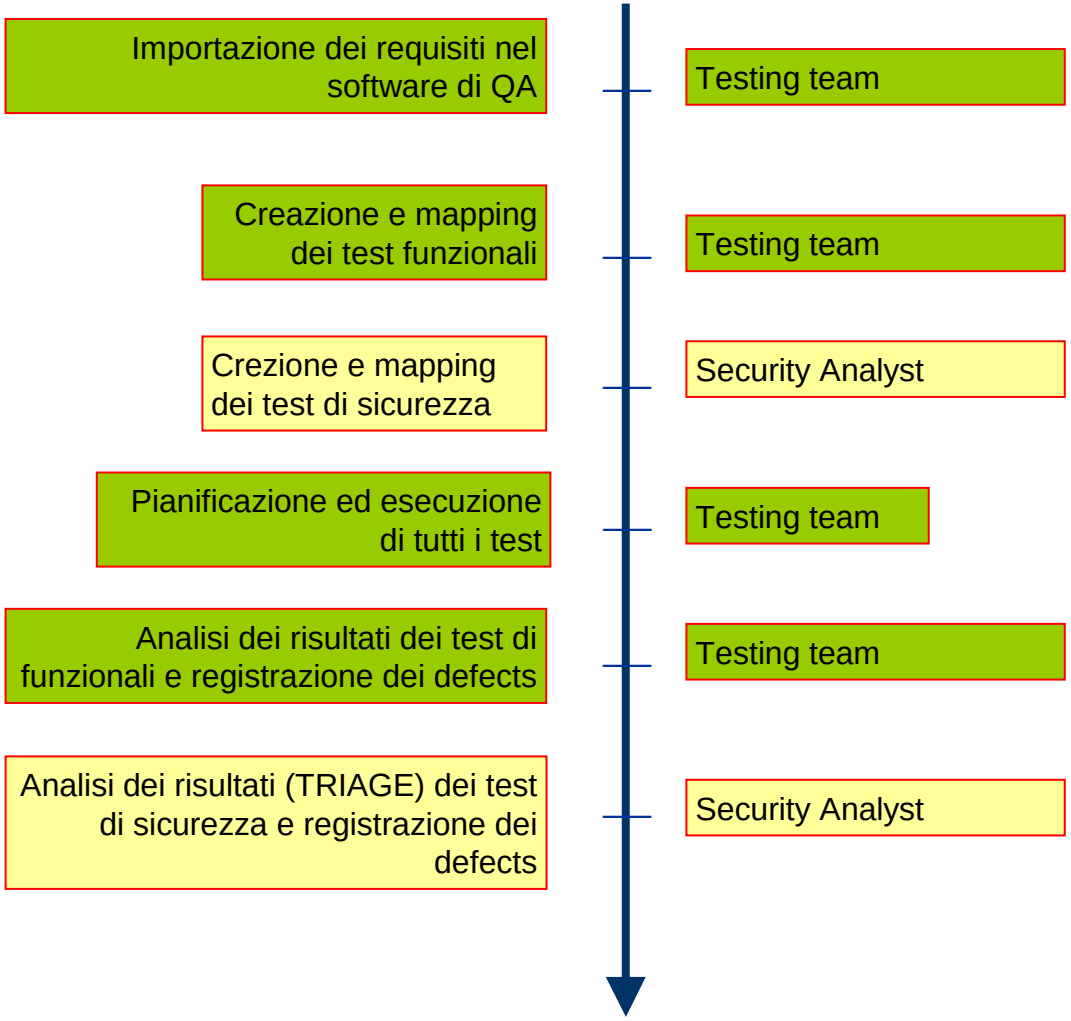
- ***Software di QA***
  - Ambiente di riferimento per la gestione dei test (tutti i tipi);
  - Realizza il mapping fra requisiti e test;
  - Registra, storicizza e segnala i defects (risultati negativi dei test).

- ***Tool per Analisi delle vulnerabilità***
  - Server per la definizione e l'esecuzione dei test di vulnerabilità;
  - Custodisce e mantiene aggiornata la security Knowledgebase.

Analisi dinamica in fase di test: architettura



Integrazione con il ciclo di vita del software



Gli Sviluppatori

- **Cosa fanno**
  - Importano le liste di Defects a loro assegnate e applicano le correzioni indicate dal Security Analyst sul progetto di riferimento seguendo il normale processo SDLC (CR,Test,Deploy,ecc);
  - Eventualmente supportano il Security Analyst nell'analisi dei risultati.

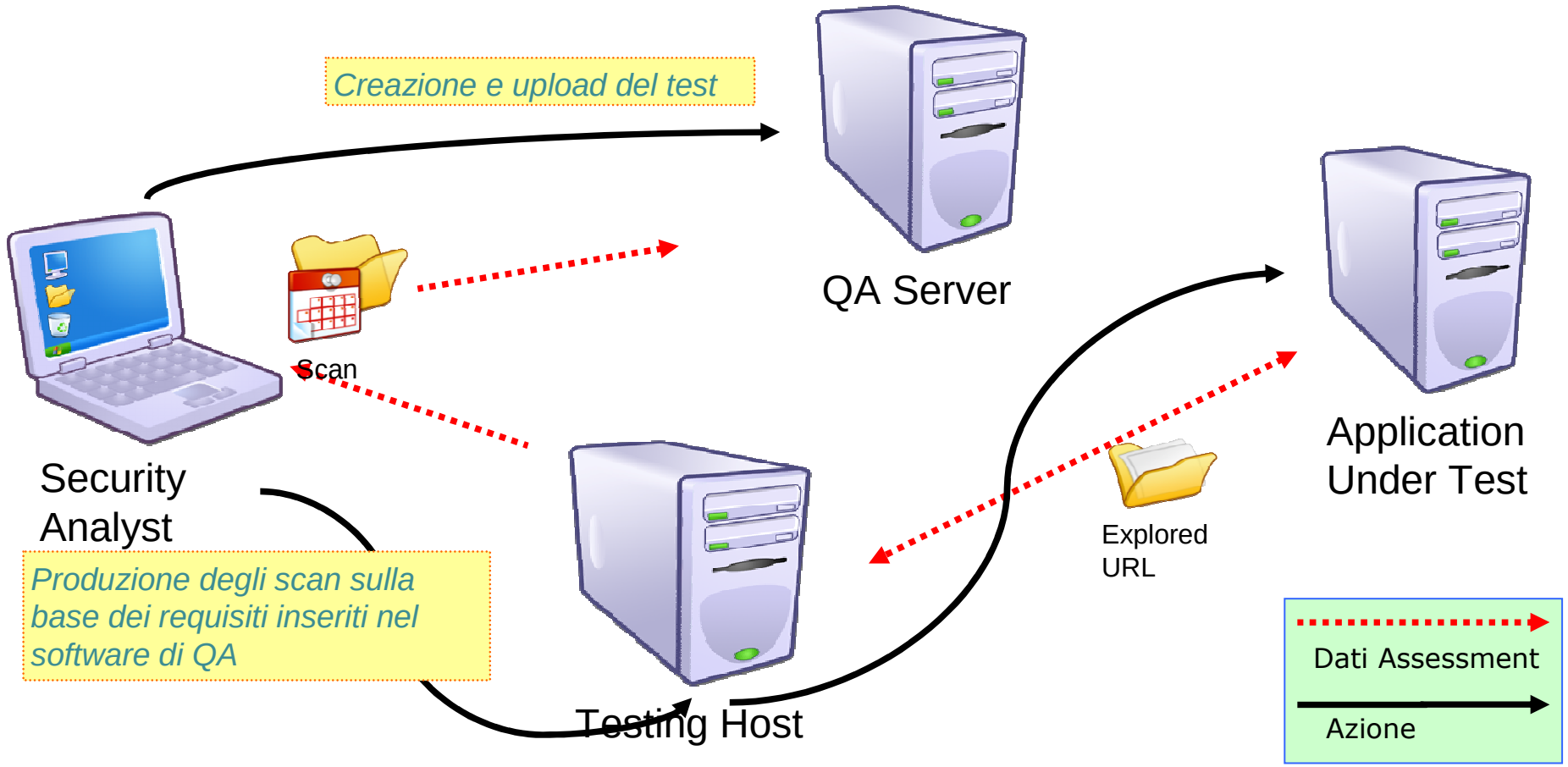
- **Cosa NON fanno**
  - Non effettuano autonomamente l'analisi dei risultati del VA.

I Security Analyst

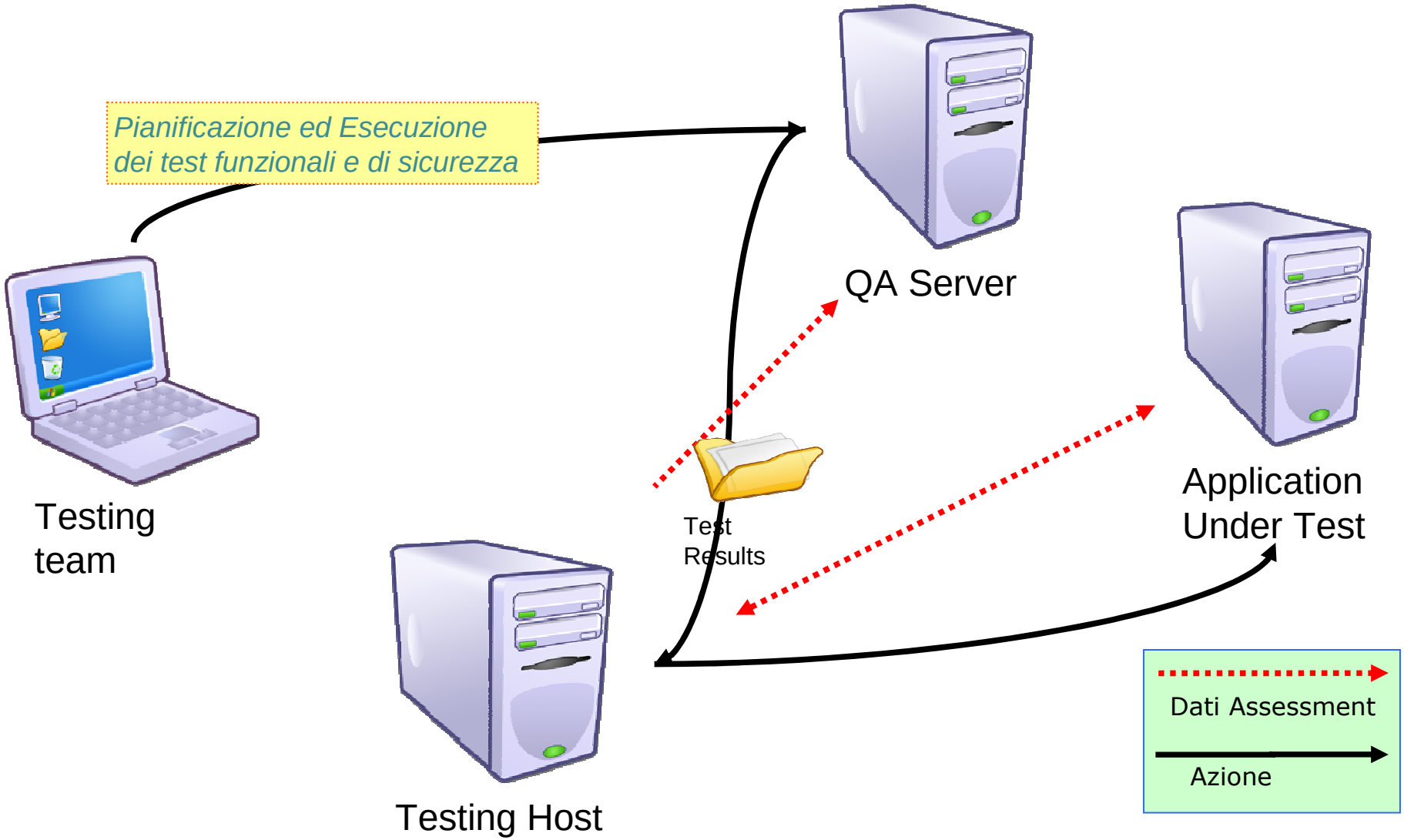
- **Cosa fanno**
  - In sincronia con la fase di test eseguono l'analisi dinamica del software ed il triage dei risultati ottenuti
  - Affiancano gli sviluppatori qualora sia necessario un supporto specifico in termini di sicurezza

- **Cosa NON fanno**
  - Non possono autonomamente modificare le policy di sicurezza: severity delle anomalie segnalate, classificazioni, ecc;
  - Non possono pianificare "assessment" utili ai fini delle metriche di sicurezza.

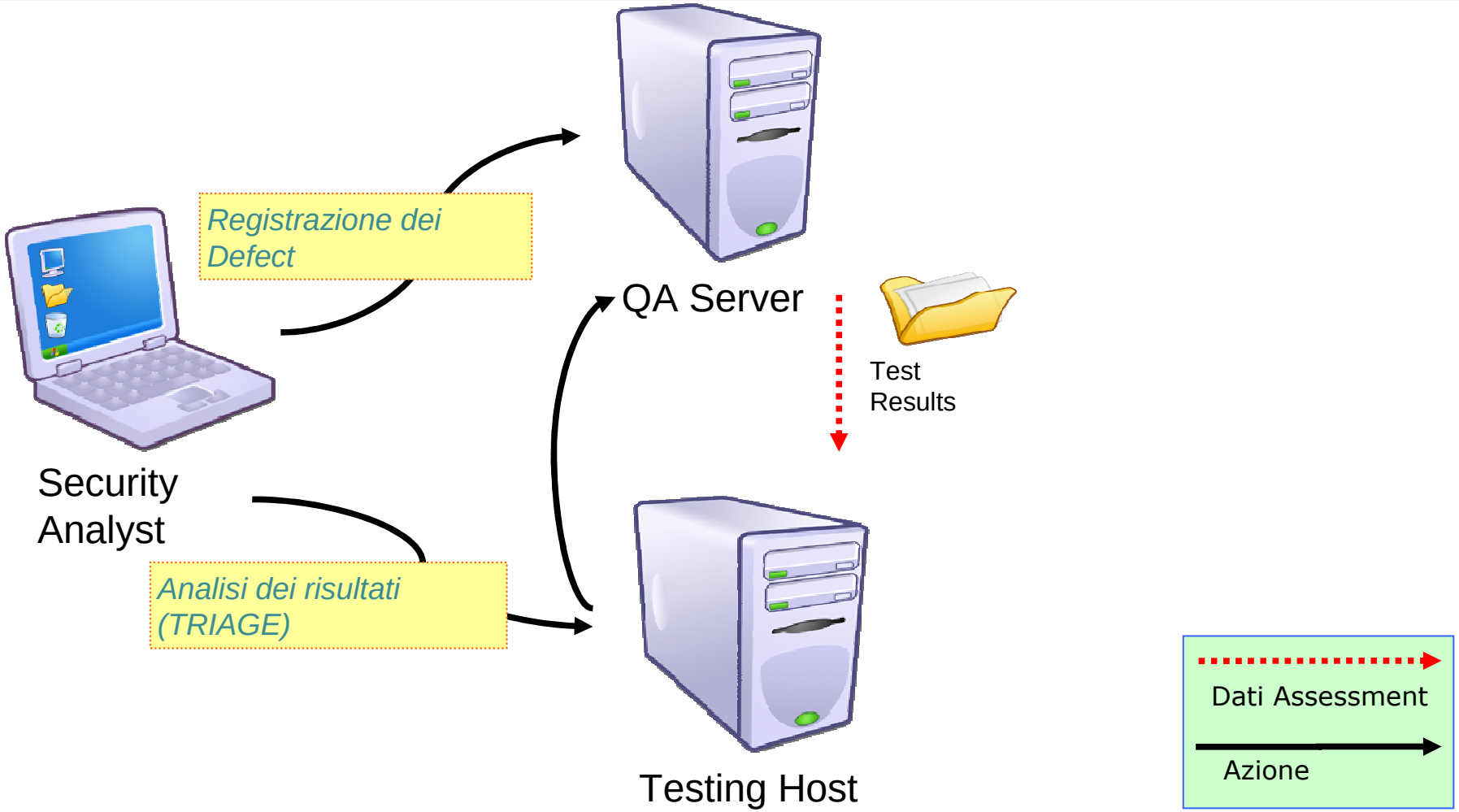
Analisi dinamica in fase di test: generazione del test



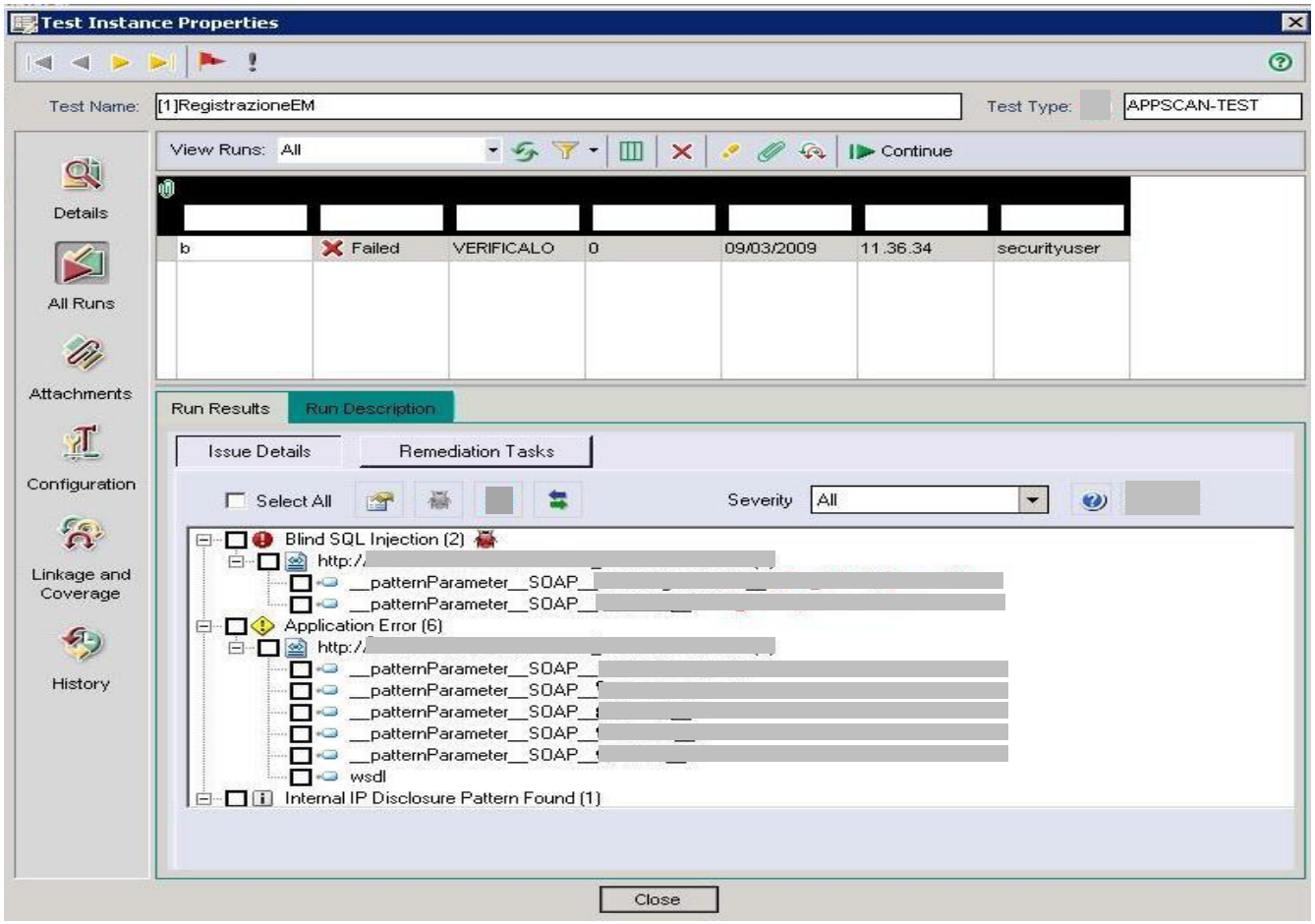
Analisi dinamica in fase di test: esecuzione



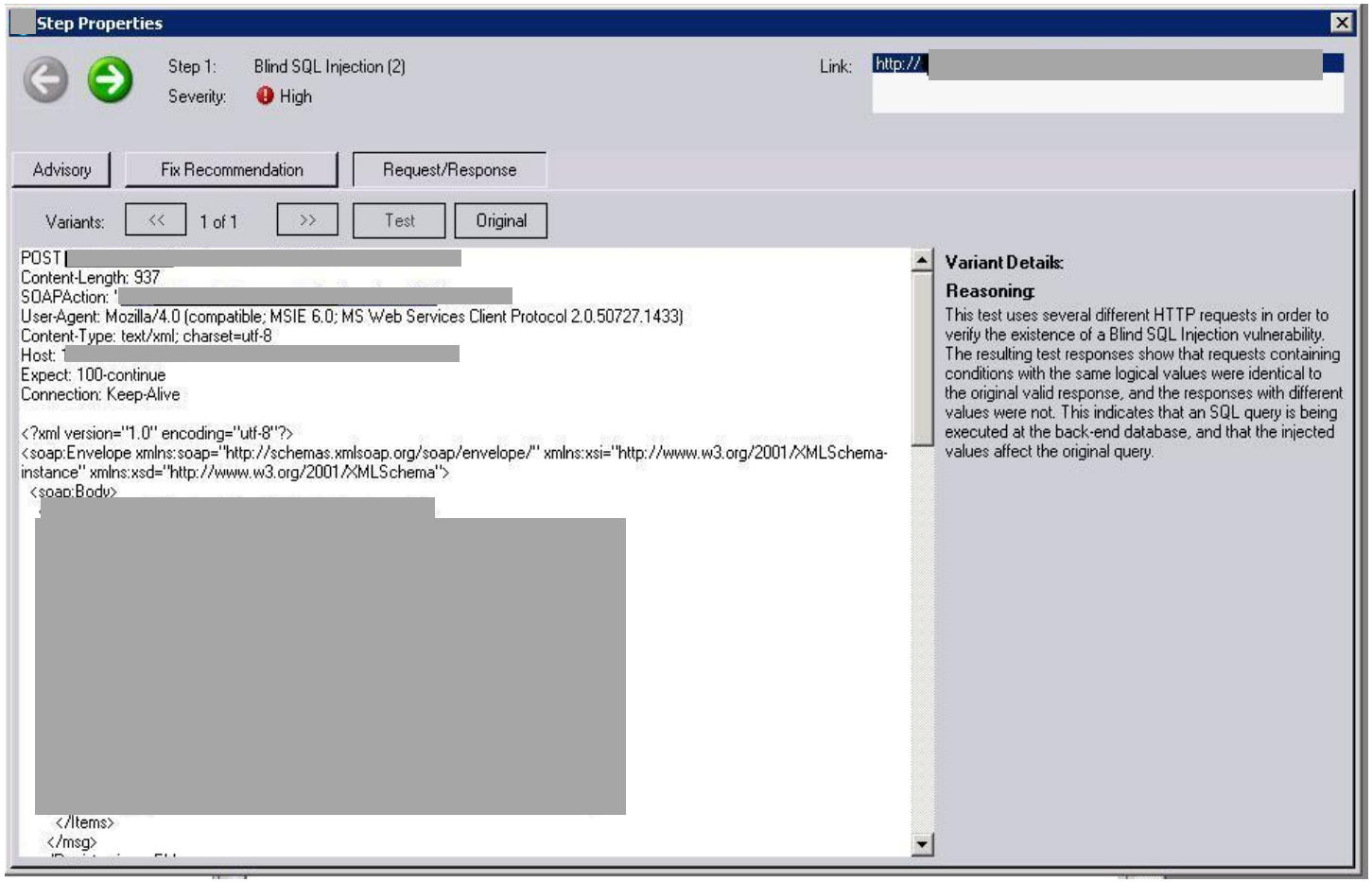
Analisi dinamica in fase di test: gestione dei risultati



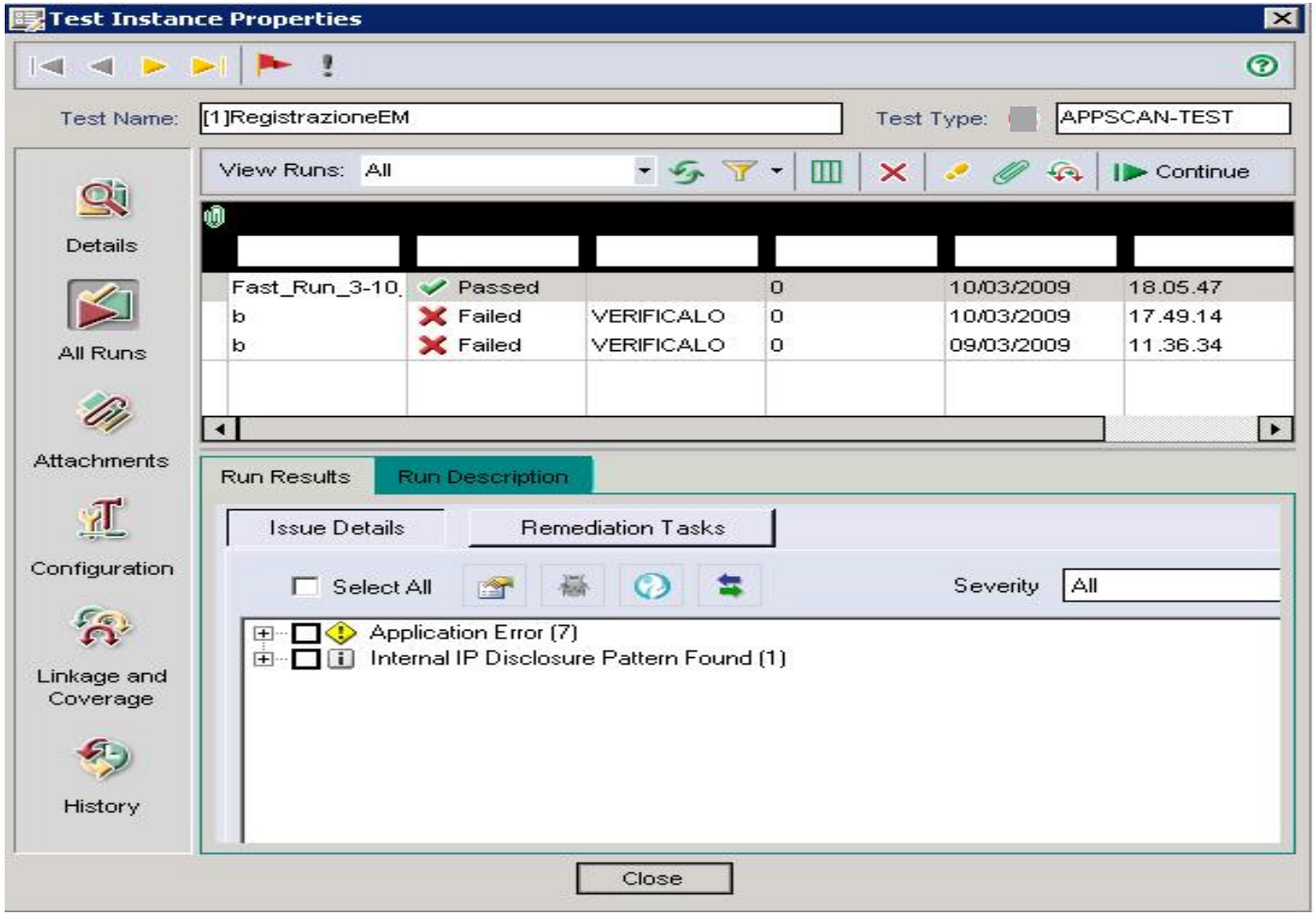
Il tool di QA: risultati



Il tool di QA: dettaglio delle vulnerabilità

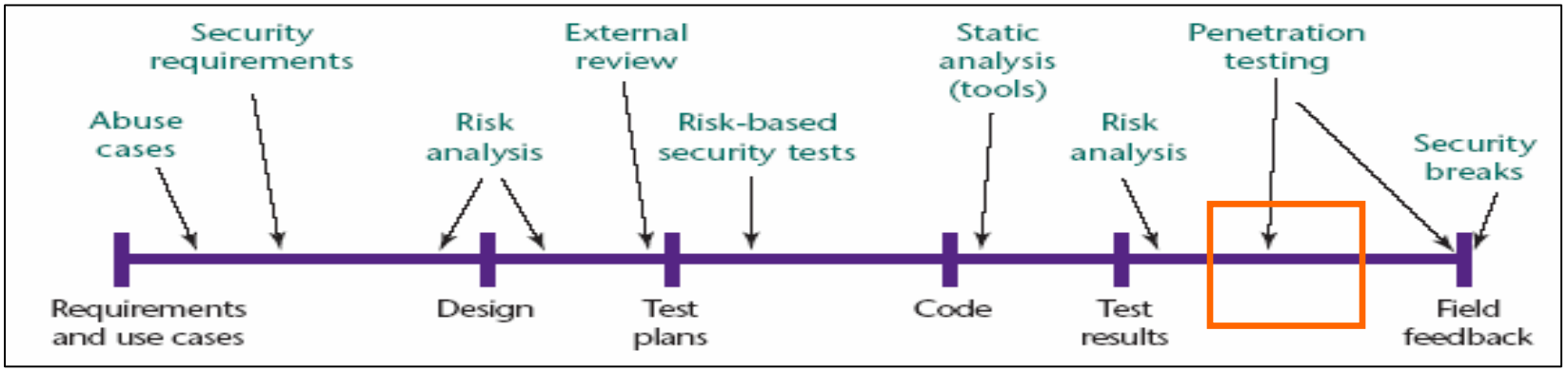


Il tool di QA: risultati a seguito dell'intervento



INDICE DELLA PRESENTAZIONE :

1. Sviluppo di software sicuro: introduzione
2. L'analisi statica del codice sorgente
3. L'analisi dinamica in fase di test
4. L'analisi dinamica delle applicazioni in produzione

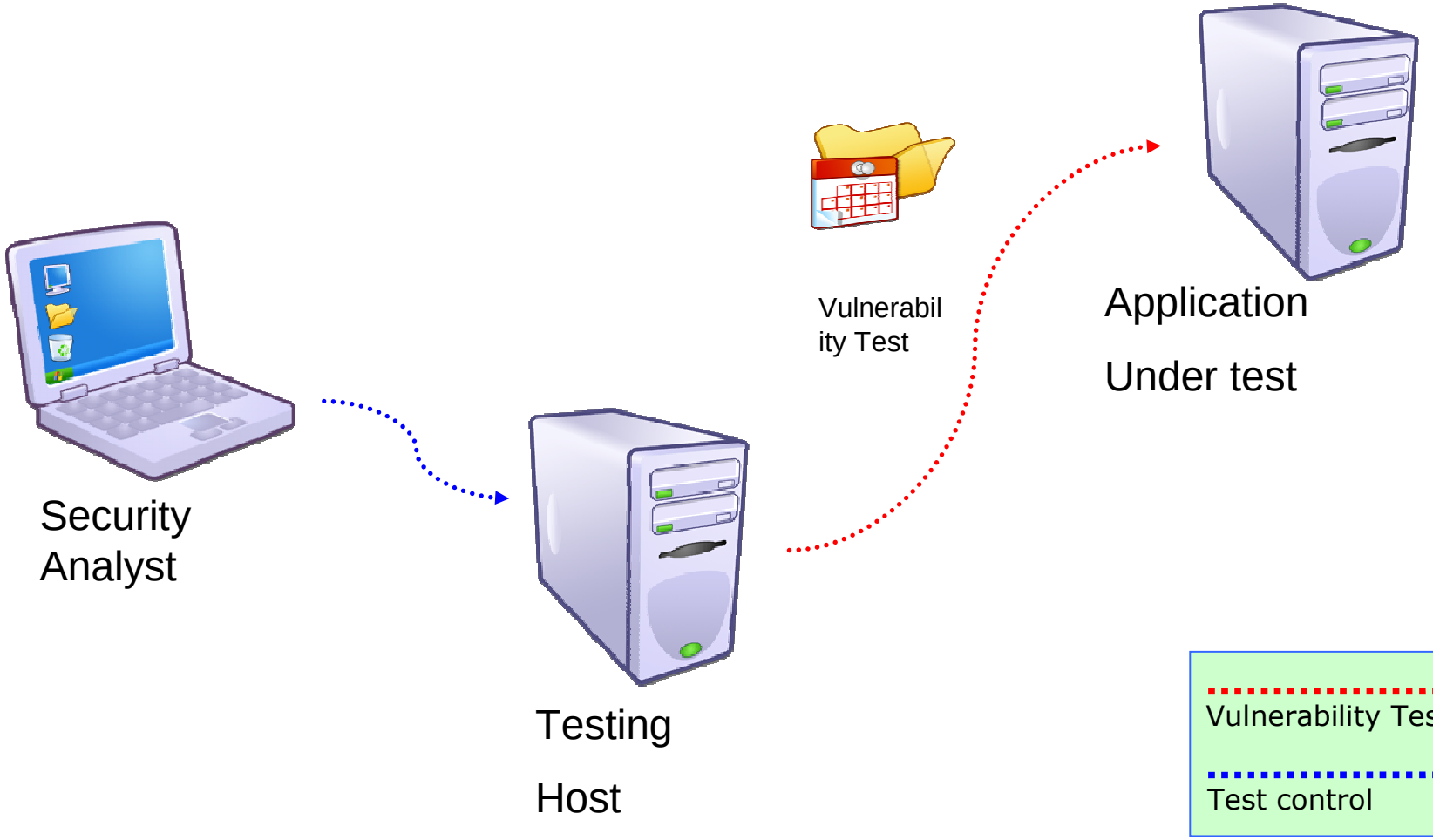

Security requirements
External review
Static analysis (tools)
Penetration testing
Abuse cases
Risk analysis
Risk-based security tests
Risk analysis
Security breaks
Requirements and use cases
Design
Test plans
Code
Test results
Field feedback

- **Impatto a livello sistemistico**
  - Nullo: l'analisi del software in fase di produzione è autonoma rispetto ai sistemi preesistenti del ciclo di vita del software.

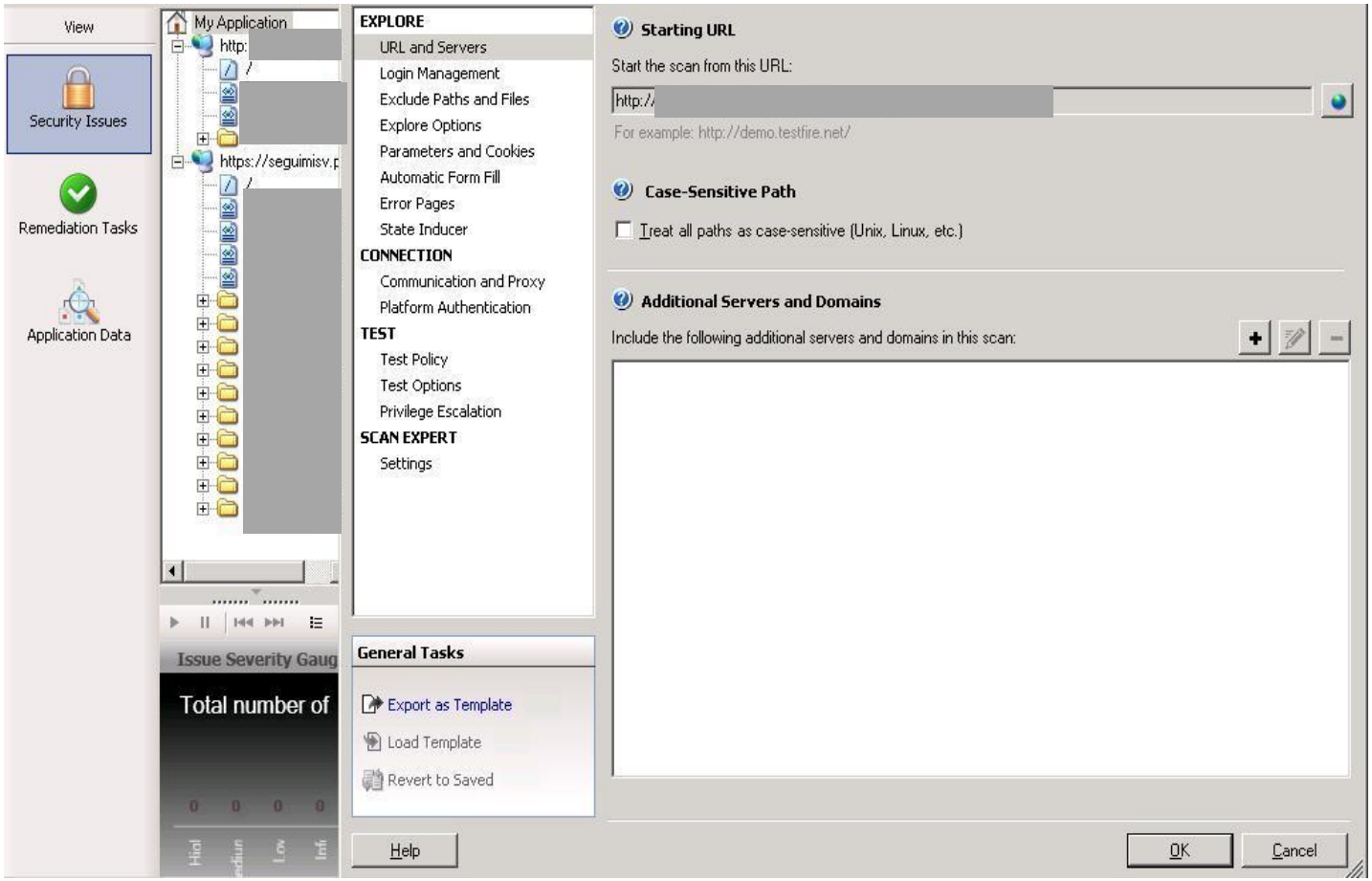
- **Impatto sui processi**
  - Esecuzione di analisi di vulnerabilità successive al rilascio in produzione del software.

- **Ruoli e competenze:**
  - Security Analyst e sviluppatori hanno competenze analoghe a quanto già visto.

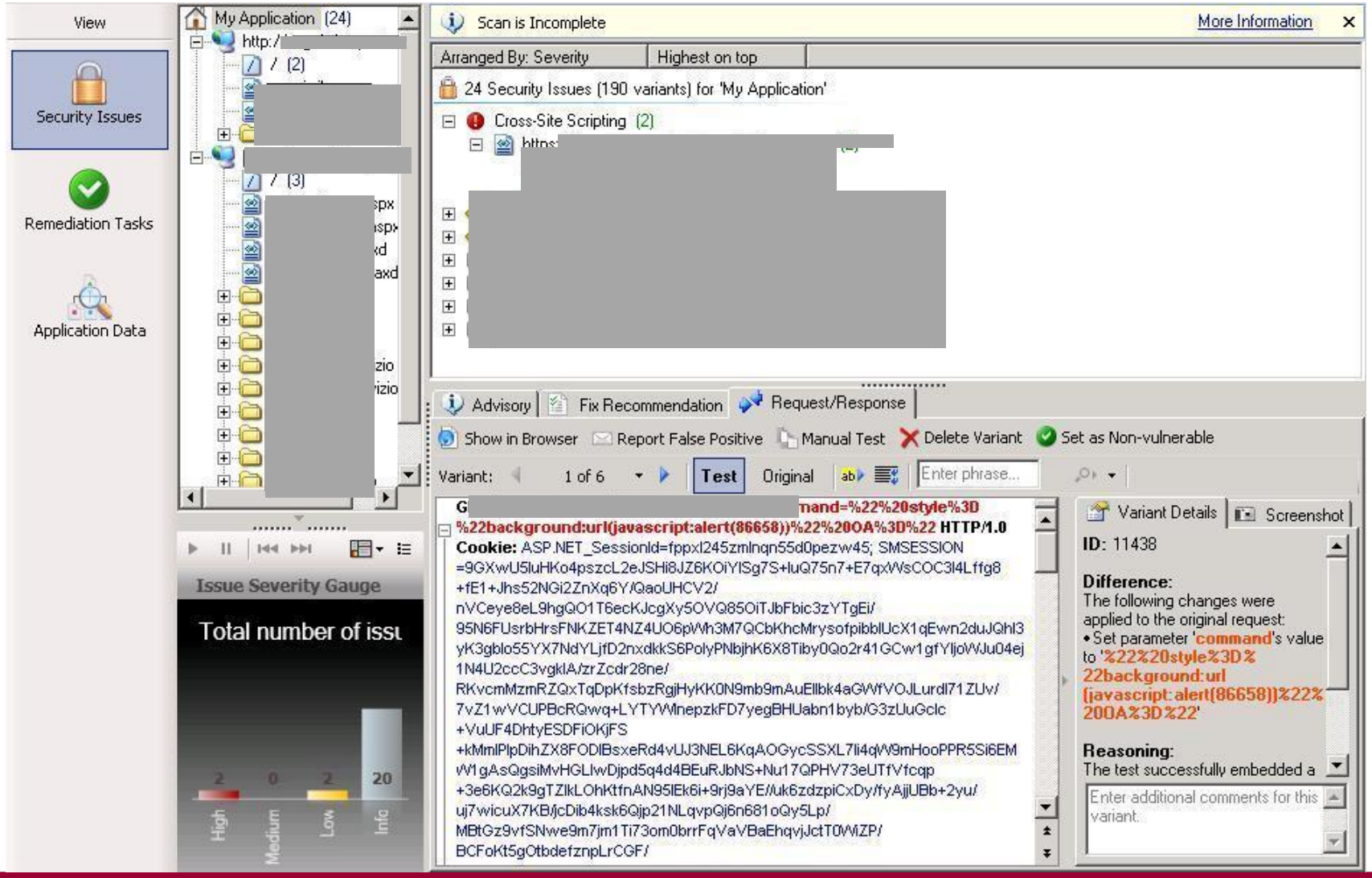
Analisi del software in produzione: architettura



Analisi del software in produzione: tool 1/2



Analisi del software in produzione: tool 2/2



Domande?

alessandro.garsia@postecom.it

26 marzo 2009