

2. Testy bezpieczeństwa

2.1. Cel przeprowadzenia testów

Testy bezpieczeństwa mają za zadanie ujawnienie ewentualnych podatności systemu na różnego rodzaju ataki np. SQL Injection, XSS. Oprócz wykrycia samych luk bezpieczeństwa, testy takie powinny również odpowiedzieć na pytanie jakie są potencjalne skutki wykorzystania odkrytych podatności oraz w jaki sposób można je wyeliminować. Odpowiedź na te pytania może ułatwić decyzję, czy z biznesowego punktu widzenia opłaca się wydawać pieniądze na poprawki mające na celu usunięcie podatności. Nie w każdej bowiem aplikacji priorytetem jest bezpieczeństwo przetwarzanych i/lub składowanych danych, można sobie wyobrazić systemy, w których wydajność jest znacznie ważniejsza od 100% bezpieczeństwa danych.

Testowana aplikacja składa się z wielu rozproszonych modułów zintegrowanych przy pomocy jednego interfejsu użytkownika. Testy aplikacji obejmowały zarówno testy pojedynczych modułów udostępniających usługi sieciowe, jak i zintegrowanej aplikacji.

Z powodu charakteru testowanej aplikacji oraz braku dostępu do serwerów, na których zostały uruchomione poszczególne moduły systemu, testy bezpieczeństwa miały charakter testów czarnoskrzynkowych (black box tests). Przed wykonaniem testów nieznane były technologie użyte do budowy poszczególnych modułów, zaimplementowane struktury danych oraz wykorzystane bazy danych. W rezultacie bardzo trudne było znalezienie odpowiedzi na pytania związane bezpośrednio z implementacją aplikacji np. czy kod aplikacji został poddany obfuskacji lub też w ilu tabelach przechowywane są dane użytkowników.

2.2. Narzędzia testujące

Z powodu braku możliwości wykorzystania do testów zamkniętego oprogramowania np. firmy IBM AppScan, do testów zostały zaprzężone darmowe narzędzie, tj. SoapUI oraz oprogramowanie Subgraph Vega.

SoapUI pozwolił na automatyczne przetestowanie udostępnianych przez poszczególne moduły usług sieciowych. Testowanie obejmowało próby wykorzystania niżej wymienionych podatności/metod:

- Cross Site Scripting (XSS)
- Invalid Types
- Malformed XML
- SQL Injection
- XML Bomb
- Xpath Injection

Zintegrowana aplikacja została przetestowana przy pomocy oprogramowania Vega, w całości zaimplementowanego w języku Java, wykorzystującego platformę Eclipse RCP jako GUI. Narzędzie zostało wzbogacone o interpreter języka javascript pozwalający na pisanie modułów rozszerzających podstawową funkcjonalność aplikacji właśnie w tym języku. Vega pozwala na testowanie m.in. takich podatności jak:

- XSS Injection
- XML Injection
- URL Injection

- Blind SQL attacks
- HTTP Header Injection attacks
- Blind XPath Injection
- Shell Injection Checks
- Format String Injections Checks
- Integer Overflow Checks

Oprócz wysyłania odpowiednio spreparowanych zapytań, narzędzie to potrafi również automatycznie sprawdzać odpowiedzi serwera w poszukiwaniu m.in.:

- Wycieków kodu aplikacji
- Hasel w postaci tekstowej
- Numerów ubezpieczeń społecznych, numerów kart kredytowych
- Adresów e-mail
- Niebezpiecznych lub nieznanych zestawów znaków (Character Sets)
- Lokalnych adresów IP
- Uwierzytelniania poprzez HTTP Basic Auth bez wykorzystania protokołu SSL/TLS

Narzędzie to potrafi również uwierzytelnić się w aplikacji wykorzystującej praktycznie dowolny sposób uwierzytelniania. Jest to możliwe dzięki funkcjonalności przechwytywania zapytań uwierzytelniających użytkownika w aplikacji (uruchomienie narzędzia w trybie proxy).

Ostatecznym etapem testów były testy manualne, polegające na sprawdzaniu typowych błędów aplikacji webowych np. zwracania w odpowiedzi stosów błędów do klienta lub też niepoprawnej implementacji obsługi przechowywanych hasel.

Implementacja testów

SoapUI

Testom zostały poddane wszystkie¹ moduły wchodzące w skład aplikacji, jednak z powodu mnogości zaimplementowanych operacji w każdym z nich, tylko dla wybranej części operacji zostały utworzone zestawy testów (ang. test suite). Przetestowane operacje z podziałem na moduły przedstawia poniższa tabela.

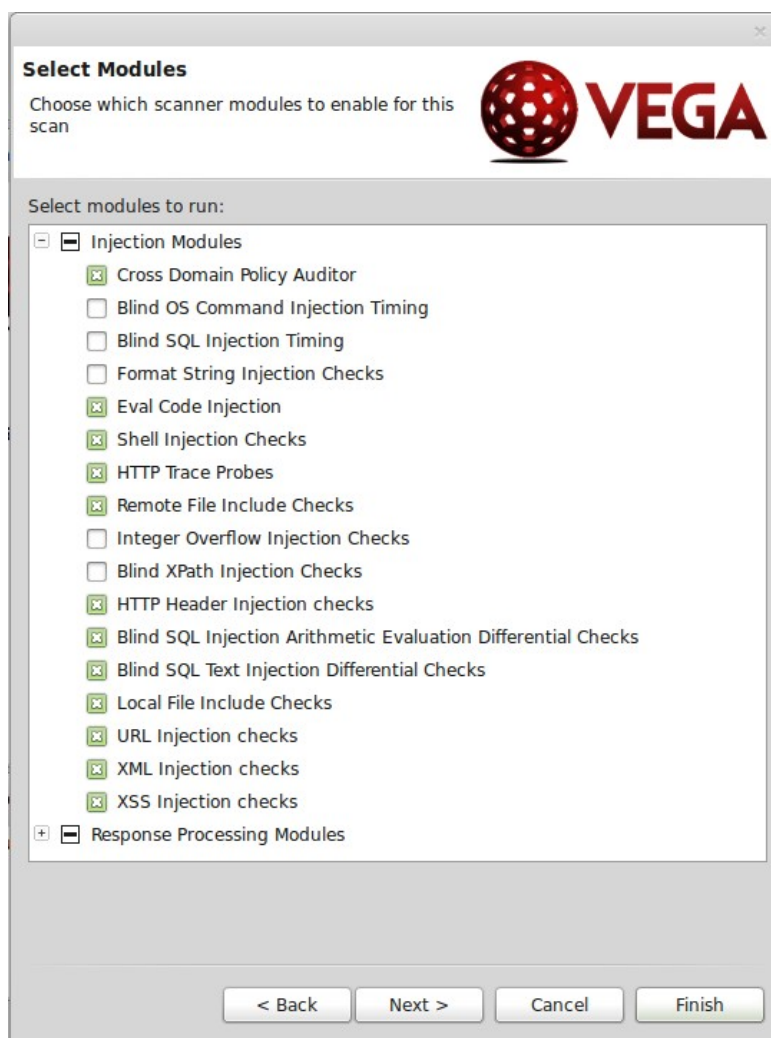
Moduł aplikacji	Testowana Operacja TODO
Panel administracyjny	isGroupAllowed
Moduł archiwizacji dokumentów	AddNewDocument
	EditDocumentDescription
Moduł zgłaszania szkody	ReportInformation
Moduł do komunikacji z rzeczoznawcami	addTicket
	takeOwnership
Moduł do obsługi workflow	addWorkflow
	executeAction

¹ Poza modulem „Panel użytkownika”, który nie został udostępniony publicznie.

Moduł aplikacji	Testowana Operacja TODO
Moduł zarządzania kontem użytkownika	authorization
	changeEmail
	register
	setUserParam

Subgraph Vega

Oprogramowanie Vega zostało użyte do przetestowania zintegrowanej aplikacji. Ponieważ większość zasobów testowanej aplikacji jest dostępna dopiero po uwierzytelnieniu się w niej (podaniu poprawnego loginu oraz hasła) konieczne było utworzenie makra, za pomocą którego wykorzystywany skaner mógł się uwierzytelnić w aplikacji. W trakcie testów został użyty standardowy zestaw parametrów zaprezentowany poniżej.



Rysunek 1: Wybrane metody ataków podczas skanowania aplikacji

Skaner został tak ustawiony, aby przeszukiwał (wykrywał) w odpowiedziach serwera m.in. następujące anomalie:

- wycieki kodu źródłowego,

- hasła w postaci „plain text”,
- uwierzytelnienie Basic HTTP bez użycia protokołów szyfrujących,
- listowanie katalogów (błędnie ustawienia w plikach .htaccess),
- wewnętrzne adresy IP,
- nieznane wartości w nagłówkach „Char-set”,

Testy manualne

„Implementacja” testów manualnych polegała na próbie odpowiedzenia na następujące pytania:

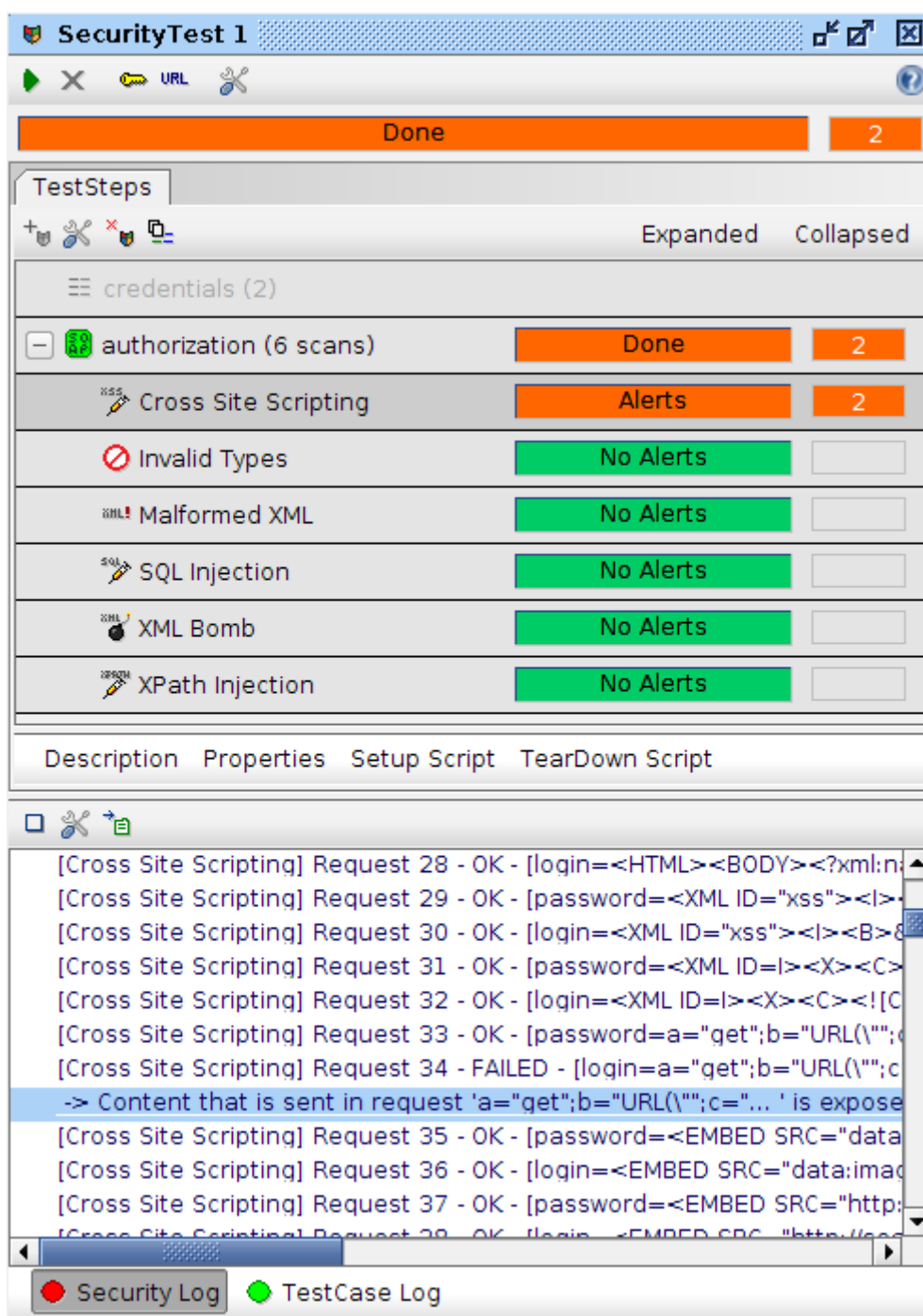
1. Czy usługi sieciowe wykorzystywane w aplikacji są udostępnione publicznie?
2. Czy w aplikacji są przechowywane / przesyłane hasła w postaci „plain text”? W jaki sposób są przechowywane hasła?
3. Czy formularz logowania / rejestracji jest poprawnie zaprojektowany (zwracane kody błędów)?
4. Czy w odpowiedziach serwer przesyła informacje ujawniające informacje takie jak użyta technologia, kody błędów itp.?
5. Czy możliwe jest wykonanie wrażliwych operacji np. usunięcie użytkownika, bez wcześniejszego uwierzytelnienia i autoryzacji?
6. Czy kod aplikacji został poddany obfuscacji przed zbudowaniem wersji binarnej?
7. Czy aplikacja weryfikuje poprawność certyfikatu SSL.
8. W jaki sposób zapewniony jest mechanizm utrzymania sesji (pliki, baza danych, ram)?
9. W jaki sposób chroniona jest przesyłana treść (SSL)?
10. Czy oprogramowanie zapisuje logi tekstowe, w których są zapisywane dane użytkownika (jakie)?
11. Czy w logach są zapisywane hasła lub skróty haseł?
12. Czy hasła mają wymuszaną długość/znaki etc.?

Rezultaty testów

SoapUI

Moduł zarządzania kontem użytkownika

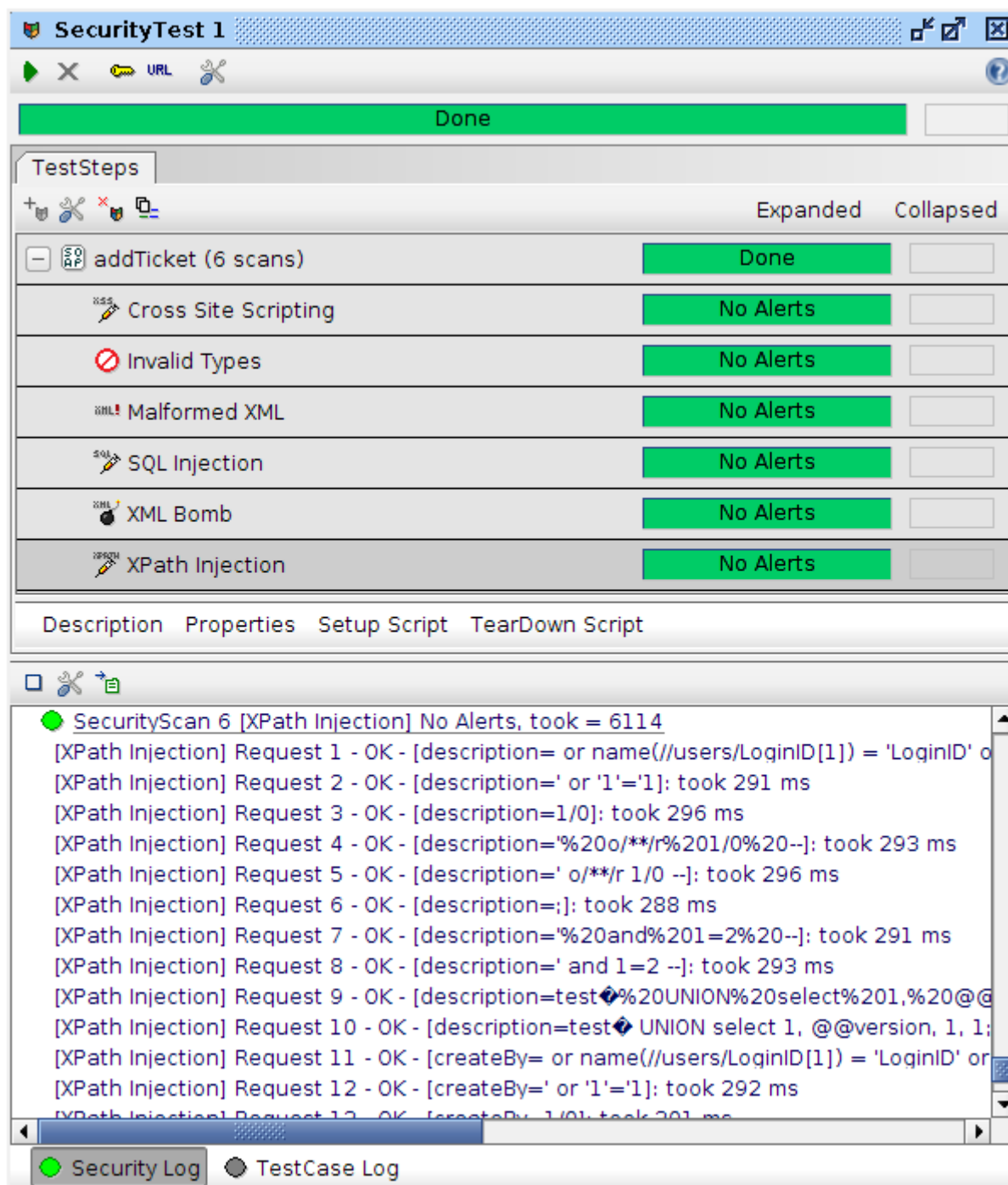
W trakcie testów zostały wykryte podatności XSS powodowane przez zwracanie w odpowiedzi tych samych danych, które zostały wysłane jako argumenty operacji. Poniżej znajduje się zrzut ekranu prezentujący wyniki skanowania operacji „authorization”.



Rysunek 2: Wyniki skanowania operacji authorization

Moduł komunikacji z rzeczoznawcami

SoapUI nie wykrył żadnych podatności w tym module.



Rysunek 3: Rezultat skanowania w przypadku braku wykrytych podatności

Panel Administracyjny

SoapUI nie wykrył żadnych podatności w tym module.

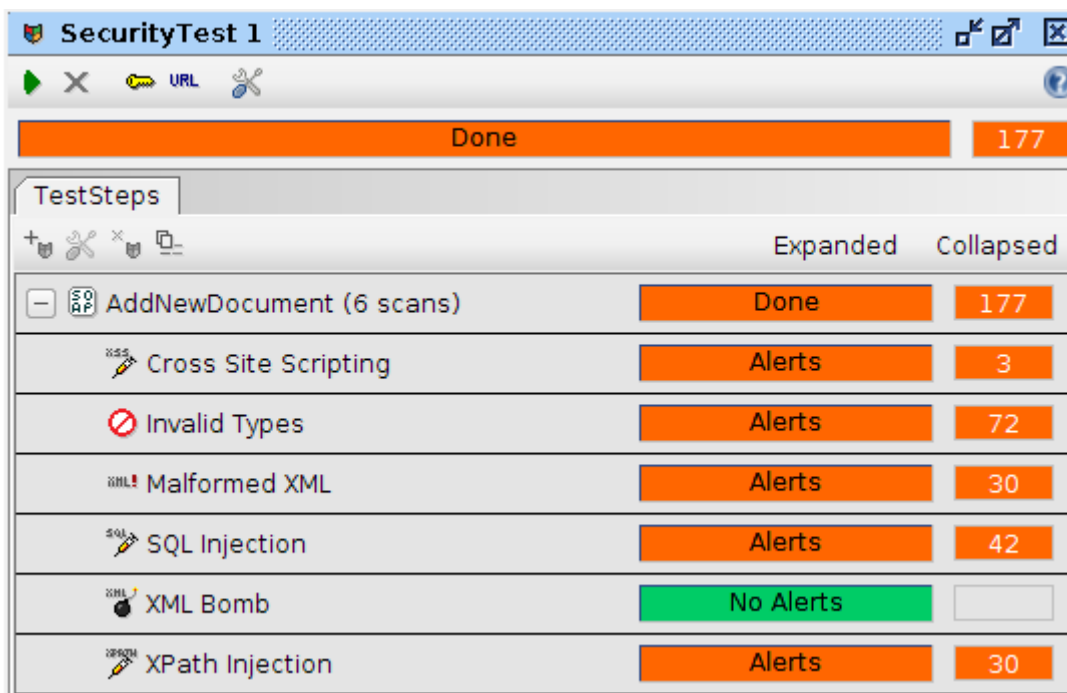
Moduł archiwizacji dokumentów

W tym module zostały wykryte dwa główne problemy:

1. Zwracanie „stack traców” w przypadku przesyłania błędnych argumentów – nie jest to dobra praktyka, gdyż przy pomocy takiego stosu błędów atakujący może z łatwością dowiedzieć się jakiej technologii używa dana usługa sieciowa oraz z jakich bibliotek

korzysta. Wiedza ta może pomóc przy poszukiwaniu błędów konkretnej biblioteki w konkretnej wersji oprogramowania

2. Podobnie jak w przypadku modułu do zarządzania kontem w tym module może zostać wykorzystana podatność typu XSS.



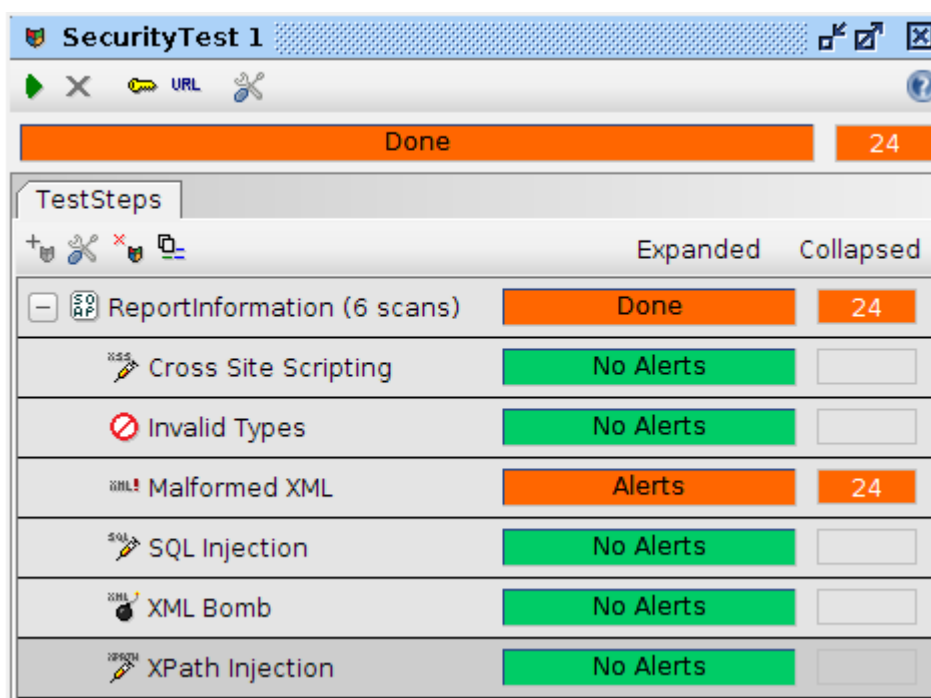
The screenshot shows the SecurityTest 1 application window. At the top, a status bar indicates 'Done' with a count of 177. Below this, the 'TestSteps' section is expanded, showing a list of scan operations and their results. The operations include Cross Site Scripting, Invalid Types, Malformed XML, SQL Injection, XML Bomb, and XPath Injection. The results are displayed in orange bars with 'Alerts' and counts.

TestStep	Expanded	Collapsed
AddNewDocument (6 scans)	Done	177
Cross Site Scripting	Alerts	3
Invalid Types	Alerts	72
Malformed XML	Alerts	30
SQL Injection	Alerts	42
XML Bomb	No Alerts	
XPath Injection	Alerts	30

Rysunek 4: Wyniki skanowania operacji AddNewDocument modułu archiwizacji dokumentów

Moduł zgłaszania szkody

Skanowanie modułu wykazało, że przy wysłaniu zdeformowanego XMLa moduł odpowiada stosem błędów. Jest to zachowanie nieodpowiednie (podobnie jak w innych modułach, gdzie zaobserwowano zwracanie stosów błędów).



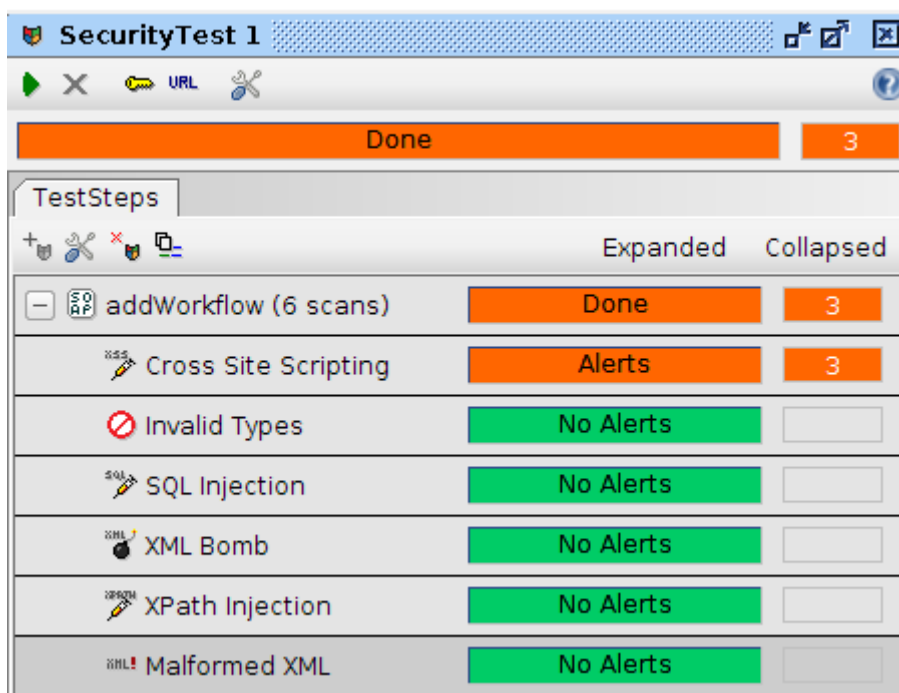
The screenshot shows the SecurityTest 1 application window. At the top, a status bar indicates 'Done' with a count of 24. Below this, the 'TestSteps' section is expanded, showing a list of scan operations and their results. The operations include Cross Site Scripting, Invalid Types, Malformed XML, SQL Injection, XML Bomb, and XPath Injection. The results are displayed in green bars with 'No Alerts' and counts.

TestStep	Expanded	Collapsed
ReportInformation (6 scans)	Done	24
Cross Site Scripting	No Alerts	
Invalid Types	No Alerts	
Malformed XML	Alerts	24
SQL Injection	No Alerts	
XML Bomb	No Alerts	
XPath Injection	No Alerts	

Rysunek 5: Wyniki skanowania operacji ReportInformation modułu zgłaszania szkody

Moduł do obsługi workflow

Testy wykazały możliwość wykorzystania podatności typu XSS w testowanym module. Innych podatności nie stwierdzono.



Rysunek 6: Wyniki skanowania operacji addWorkflow modułu do obsługi workflow

Subgraph Vega

Poniżej zostało przedstawione podsumowanie przeprowadzonego skanu systemu.

Scan Alert Summary

High		(None found)
Medium		(1 found)
Possible Source Code Disclosure	1	
Low		(1 found)
ASP/ASPX Error Detected	1	
Info		(350 found)
Character Set Not Specified	1	
Interesting Meta Tags Detected	349	

Rysunek 7: Wyniki skanowania aplikacji przy użyciu narzędzia Subgraph Vega

Skanowanie nie wykryło żadnych poważnych podatności aplikacji. Wykryta podatność na poziomie „Medium” okazała się być pomyłką skanera – wyrażenie regularne napisane w języku JavaScript

zostało potraktowane jako wyciek kodu strony ASP lub JSP. Pozostałe zgłoszenia dotyczyły w znakomitej większości meta tagów nagłówek odpowiedzi, w których występowały takie słowa jak „Windows”, „Linux”, co skaner potraktował jako możliwość wycieku informacji opisujących wykorzystywaną przez aplikację platformę (system operacyjny). W jednym przypadku okazało się, że zestaw znaków nie został poprawnie określony, co jednak nie jest problemem w kontekście bezpieczeństwa aplikacji.

Podsumowując skan nie wykazał żadnych podatności aplikacji, co można uznać za spory sukces.

Testy manualne

1. Usługi sieciowe wykorzystywane w aplikacji są dostępne publicznie. Powoduje to stanowiące zwiększenie liczby potencjalnych miejsc w systemie, które mogą posiadać podatności pozwalające na nieautoryzowany dostęp do zasobów aplikacji.
2. Hasła są przesyłane w postaci „plain text” (wynika to z braku zastosowania protokołów szyfrujących np. SSL/TLS). Nie stwierdzono natomiast przechowywania haseł w postaci plain text w bazie danych aplikacji.
3. Operacja rejestracji nie jest zaprojektowana poprawnie, ponieważ ogranicza maksymalną długość hasła użytkownika do 16 znaków. Za plus można uznać wymuszanie na użytkownikach wymyślanie haseł dłuższych niż 8 znaków. Operacja „logowanie” nie jest zaprojektowana poprawnie, ponieważ bazując na odpowiedzi przesyłanej z serwera wiadomo, czy login czy hasło było błędne. Taka „podpowiedź” może znacząco ułatwić ataki typu brute force.
4. Tak, przynajmniej część modułów w przypadku celowo spreparowanych, błędnych argumentów przesyła w odpowiedzi stosy błędów ujawniające z jakiej technologii korzysta dany moduł, z jakich bibliotek itp. Takie informacje mogą nakierować atakującego na poszukiwanie błędów w implementacjach konkretnych serwerów http, serwerów aplikacji, frameworkach itp.
5. Tak, moduły aplikacji np. „Moduł zarządzania kontem użytkownika” pozwalają na wykonywanie wrażliwych operacji (np. usunięcie użytkownika) bez konieczności wcześniejszej autoryzacji.
6. Bez dostępu do kodu źródłowego i procesu budowania binariów nie jest możliwe w 100% stwierdzenie, czy kod został poddany obfuskacji. Jednak na podstawie stosów błędów możliwe jest, z dużą dozą prawdopodobieństwa stwierdzenie, że kod **nie** został poddany obfuskacji – wskazują na to „przyjazne” programiście nazwy metod oraz zmiennych.
7. W żadnym module aplikacji nie stwierdzono użycia szyfrowanych połączeń z wykorzystaniem protokołów SSL/TLS.
8. Do utrzymywania sesji aplikacja używa plików cookie (ciasteczek). Token przechowywany w ciasteczkach jest najprawdopodobniej generowany na podstawie danych użytkownika oraz stempla czasowego. Metoda taka uznawana jest za względnie bezpieczną.
9. Przesyłana treść nie jest chroniona, tj. nie są używane żadne protokoły szyfrujące dane.
10. Tak, aplikacja zapisuje dane użytkownika oraz operacje takie jak:
 - stary oraz nowy adres e-mail podczas zmieniania adresu
 - login użytkownika przy okazji wykonywania większości operacji
11. Nie.
12. Aplikacja wymusza jedynie minimalną długość hasła (8 znaków). Niestety nie jest sprawdzana złożoność haseł, tj. nie jest wymagane, aby hasła zawierały liczby, wielkie oraz

małe litery, znaki specjalne. Nie jest również sprawdzane, czy użytkownik nie oparł swojego hasła na danych, które podał w formularzu rejestracyjnym (login, adres e-mail).

Wnioski i zalecenia

Na podstawie wyników uzyskanych w testach bezpieczeństwa można dojść do wniosku, że w obecnej formie system nie nadaje się do udostępnienia go użytkownikom.

Niedopuszczalne jest, aby było możliwe wykonanie takich operacji jak usuwanie konta, czy adresu e-mail użytkownika bez żadnej wcześniejszej autoryzacji. Aplikacja zawiera również szereg innych błędów, takich jak np. nieprawidłowo zaprojektowana logika rejestracji/logowania użytkowników, czy zwracanie informacji pozwalających na wywnioskowanie z jakich technologii korzysta aplikacja. Na plus zasługuje odporność zintegrowanej aplikacji na popularne ataki np. XSS bądź SQL Injection. Nie zmienia to jednak faktu, że obecna architektura (publicznie dostępne API modułów) absolutnie przekreślają możliwość produkcyjnego użytkowania systemu.

Co więcej, rezultaty testów wydajnościowych dowodzą, że o ile istotną cechą aplikacji jest jej niezawodność oraz bezawaryjność w przypadku zwiększonej ilości użytkowników pracujących równolegle, również na tej płaszczyźnie nie może być mowy o systemie „bezpiecznym”.

Autorzy niniejszego raportu zalecają, aby wszystkie moduły systemu zostały przeniesione do niedostępnej publicznie sieci LAN. Wówczas po przeprojektowaniu wskazanych wcześniej operacji, ponownym przeprowadzeniu testów bezpieczeństwa oraz zwiększeniu wydajności systemu będzie można rozpocząć produkcyjne eksploataowanie systemu.