Triaging the Enterprise for Application Security Assessments

Rebecca Deck

Triaging the Enterprise for Application Security Assessments

GIAC (GCCC) Gold Certification

Author: Rebecca Deck, sdsecurityacct@hotmail.com
Advisor: Adam Kliarsky
Accepted: October 27, 2016

Abstract

Conducting a full array of security tests on all applications in an enterprise may be infeasible due to both time and cost. According to the Center for Internet Security, the purpose of application specific and penetration testing is to discover previously unknown vulnerabilities and security gaps within the enterprise. These activities are only warranted after an organization attains significant security maturity, which results in a large backlog of systems that need testing. When organizations finally undertake the efforts of penetration testing and application security, it can be difficult to choose where to begin. Computing environments are often filled with hundreds or thousands of different systems to test and each test can be long and costly. At this point in the testing process, little information is available about an application beyond the computers involved, the owners, data classification, and the extent to which the system is exposed. With so few variables, many systems are likely to have equal priority. This paper suggests a battery of technical checks that testers can quickly perform to stratify the vast array of applications that exist in the enterprise ecosystem. This process allows the security team to focus efforts on the riskiest systems first.

1. Introduction

In mature enterprises, application security and penetration testing programs exist to find vulnerabilities in internally developed applications and the complex interactions between systems (Scarfone et al., 2008). Both programs should be integrated with the Secure Development Lifecycle (SDL) to prevent vulnerabilities in internally developed applications from reaching the end users (Conklin & Shoemaker, 2014). Even commercial and third-party developed systems still warrant some steps of this process. Performing in-depth security assessments of all systems in an enterprise is, unfortunately, a long and costly undertaking (Scarfone et al., 2008). During this lengthy process, it is possible that some systems that security testers will not test applications in an order commensurate with the risk to an organization. This paper covers some of the shortcomings with current prioritization methods and proposes an alternative scheme to overcome these limitations.

Application security is a key part of a "defense in depth" strategy. This control is often only considered for internally developed software, but attackers look for vulnerabilities in all systems (McGraw, 2006). While this is true for several of the measures in the Application Software Security control, this control is more extensive than basic testing of in-house created applications. The Critical Security Controls (CSC) advise that vendors must support all software, all systems must be behind a protocol-aware firewall, system owners must maintain a development environment that is separate from production, and harden all database servers. The controls also advise in-depth testing of any application that is created by an in-house development team or by a third party explicitly for an organization (Center for Internet Security, 2016). The importance of application-specific firewalls cannot be over-estimated. These tools are instrumental in mitigating vulnerabilities discovered during application security and penetration testing until patches are available.

A penetration test involves an experienced information security practitioner attacking a target network using the same techniques as real attackers. Penetration testing differs from other methods of finding vulnerabilities in that testers exploit vulnerabilities

to access sensitive data or specific targets (Northcutt et al., 2006). The purpose of this exploitation is to illustrate the actual risk to the organization resulting from security vulnerabilities. In the context of the Critical Security Controls, red teaming is a specialized type of penetration test where the testers emulate the tactics, techniques, and procedures of a specific adversary. System owners often fail to understand how their opponents operate, which makes it difficult to prioritize security efforts (Peake, 2003). These testing activities also aid in discovering how the compromise of a system with a low data classification can be used to gain access to more sensitive systems in unexpected ways (Center for Internet Security, 2016).

This paper does not describe the methods by which one conducts application security or penetration tests. Each of these topics already has volumes of material written with descriptions of vast numbers of specific tests that an analyst must perform. The critical controls contain several of the basic qualities that are necessary for application security and penetration testing programs. For a more expansive list of activities, the Building Security in Maturity Model (BSIMM) contains a thorough listing of activities that organizations should perform as part of testing in the SDL including threat modeling, static analysis, dynamic analysis tools and penetration testing (Merkow & Raghavan, 2010). Penetration testers have several exceptional choices when determining specific test cases that analysts should run on in-scope systems. For general penetration testing, the Penetration Testing Execution Standard (PTES) covers hundreds of specific checks that may be conducted based on the scope of a test. If the test includes a web application, the Open Web Application Security Project (OWASP) produces a testing guide that checks for many common web application misconfigurations and vulnerabilities (Meucci & Mueller, 2014). The testing team should select a methodology and series of tests that are appropriate for each type of system within the enterprise.

1.1. Penetration Testing and AppSec Prioritization

Penetration testing and securing custom applications, while important, should not be the highest priority task when securing the enterprise. The Center for Internet Security defines the Critical Security Controls for Effective Cyber Defense to assist organizations in determining the order in which to implement some of the National Institute of

Standards and Technology (NIST) 800-53 priority one controls. Based on the guidance in version 6.1 of these controls, application security is the 18th control family and penetration testing is the 20th, and final, control family. The controls also explicitly state that penetration testing and red teaming do not provide a significant benefit without prior work in other areas (Center for Internet Security, 2016).

One of the primary reasons for the lower ranking for the application security and penetration testing controls is that lack of these controls is not a predominant factor in modern breaches. Most recent breaches are not the result of zero-day exploits or in-depth research on an organization's custom written applications (Northcutt et al, 2006; Bing, 2016). According to Verizon, in 2015 the median amount of time a patch was available before exploitation began was 30 days. Attackers are likely to use well-known and tested vulnerabilities in attacks as long as the techniques are still effective (Verizon, 2016). If known vulnerabilities and configuration errors exist, the control families for secure configurations and continuous vulnerability assessment and remediation should address the issues. By fixing the easily found security issues first, organizations will limit successful attacks to all but the most skilled and targeted attacks.

The controls of application security and penetration testing are extremely resource intensive (NIST, 2013). Many application security tools such as static and dynamic application security testing software have substantial false positive or false negative rates (Merkow & Raghavan, 2010). These tools are useless without experienced personnel to validate whether or not findings are legitimate. Due to the manual nature of penetration testing, it does not suffer from the same false positive issues but does require a large time investment. By first implementing an effective vulnerability management and remediation program and applying secure configurations to systems before penetration testing, the time required for penetration testing will be far less. Properly implementing higher priority critical controls allows penetration testing and application security efforts to focus on more complex security issues.

After implementing the prerequisite control families, application security and penetration testing should reveal vulnerabilities that do not currently have patches. Since the lack of patches makes fully remediating these vulnerabilities impractical,

organizations are forced to consider mitigating controls. With proper application of application-specific firewalls, packet filtering firewalls, and authentication controls it is possible to reduce the exposure of any vulnerabilities discovered as a result of security testing. Without these controls, risk mitigation is rarely an option. The organization will be forced to either accept, transfer, or avoid the risk. (Conrad et al., 2010).

There are a few exceptions to the rule that organizations should save penetration testing until late in the enterprise security model. The first, and most common, reason is that regulations – such as PCI-DSS and FFIEC – mandate this activity. Penetration testing can also be useful if a security organization is having difficulty garnering support from executive management. The fact that a third party is involved and notes a risk can provide the justification for appropriating funds to enhance security. The final reason to conduct a penetration test or red team assessment is that the security team is unsure what controls to implement next. Unless an organization meets one of these three criteria, a penetration test is not warranted (J. Tarala, personal communication, September 25, 2016).

1.2. Prioritization Side-effects

Since there are so many activities to perform before application or penetration testing of systems is warranted, there will already be a mature computing environment with a vast array of applications. The organization must then determine in which order to apply various application security and penetration testing activities. Common sense dictates that organizations should first examine the applications that pose the greatest risk to the organization.

NIST suggests that either a qualitative or quantitative risk assessment process should be used to rank systems for security testing. NIST provides guidance on how to conduct a risk assessment but does not provide specific tests that should be part of this assessment (NIST, 2012). One qualitative approach is to ask a few high-level questions to discern the risk of a system. Some of the more pertinent questions are data classification, exposure to the internet, and user population (ard3n7, 2013). This methodology allows resources with no technical abilities to rank systems in a very short period. Assessors accomplish this with a combination of interviews and short surveys with system owners.

Unfortunately, a qualitative approach to risk assessment results with a few rating classifications that have many systems in each classification. In practice, the product of such a rating is even worse, as many systems will have either the highest or lowest risk rating. Using the previously mentioned qualitative methodology, all systems with sensitive data, customer data, or that are internet facing will receive the highest risk categorization (ard3n7, 2013). These factors make the qualitative approach insufficient for the purpose of deciding where to begin testing.

2. Application Security Report Cards

2.1. Overview

Josh Wright introduced the concept of using a "Report Card" format to outline a hybrid quantitative-qualitative risk assessment technique (Wright, 2015). By examining a small number of easily observable security measures, a moderately skilled analyst can arrive at a numeric score to determine how bad a given application is likely to be from a security perspective. The technical tests contained in the report cards require only a short time to perform and require only moderate technical expertise. With this methodology, it is feasible for a single analyst to examine several applications per day. In contrast, fully testing the same number of applications manually would take weeks to months. Purchasing and configuring automated tools for static or dynamic analysis also carries a cost in both time and capital.

Report cards in no way replace full penetration testing or any application security tests. Penetration testing and application security testing are designed to catalog vulnerabilities so information technology teams can address the findings. For report cards, the intent is not to deliver the card to a development team so they can remediate specific findings to achieve a higher score. These cards should be used only to prioritize applications for full testing. This situation is likely to occur when an organization first begins examining the security of applications and during surges of project deliveries.

The scoring mechanism of the report cards is designed to detect whether or not an application follows good security practices. The report card process assesses various

observable properties of an application to produce a quantitative score on a one-hundred point scale. The weighting of each test in the report card is based either on the likelihood that a component will introduce a vulnerability or ignores a common security practice. The wording of questions is such that a higher number of points indicates better security practices. Analysts may award partial credit if a test's requirements are not completely satisfied. This credit is awarded at the discretion of the analyst performing the assessment.

Tests used in the report card process are not necessarily meant to detect the most severe vulnerabilities that may be present in an application. Modern systems will often have at least some measure of protection against common vulnerabilities such as the OWASP Top 10. It is infeasible to ensure that an application is free of these vulnerabilities without extensive testing. Instead, the purpose is to determine if developers followed good security practices by inspecting easily observable properties of an application. Because each category of application will have different security properties, several different report cards are necessary. Josh Wright provided mobile application scorecards for both Android and Apple iOS applications (Wright, 2015). Due to the prevalence of Web-based applications, this paper expands the report card concept to cover these additional categories. The proposed report cards are available at https://github.com/rangercha/AppReportCards.

The report cards developed as part of this project use two categories of tests: administratively observable and dynamically observable. Administratively observable tests are best performed by using administrative access to in-scope systems and viewing the system configuration. Analysts perform the dynamic tests in much the same way as a traditional test. However, the purpose of performing these tests is not to develop exploits or to find all occurrences of a single type of vulnerability. The goal is only to determine whether or not developers appear to have included security controls or use dangerous functionality in an application.

2.2. Report Card Development Process

The author developed the web application cards by performing a series of application assessments and analyzing the results. The applications are a combination of

commercially purchased and internally developed applications in a variety of languages. After conducting a full application assessment on each application, testers reviewed the findings for easily found vulnerabilities or configurations that exacerbated vulnerabilities. Several security issues, such as command injection, are not included because they proved difficult for entry-level testers to detect reliably. Maximum point values designate how likely a test is to indicate a problem and how derelict an application is for not meeting a specific standard. For example, using a non-privileged service account is a common security practice and occurred in most applications tested. In the application that failed to adhere to this practice, it exacerbated several security vulnerabilities. Therefore, using a non-privileged service account has a high maximum point value. In contrast, performing password resets using an out-of-band system has a low point value because several acceptable in-band methods for password resets exist. However, several of the test applications that implemented password reset self-service had vulnerabilities in this functionality.

3. Report Card Integration

Report cards are only a small part of an organization's security program. The purpose of the report cards is only to help prioritize systems for undertaking critical controls 18 and 20. To effectively use the cards, organizations should first implement all previous controls, plan how to implement the critical controls of application security and penetration testing, prioritize systems, and execute full test plans.

3.1. Phase 1 – Previous Controls

Beginning application security or penetration testing when an organization's security program is still immature will not be an effective use of assets. By first implementing prior controls from the CIS Critical Security Controls, application security and penetration testing will be far more beneficial. Security tests will be able to focus more on complex or custom vulnerabilities in a network. Additionally, the organization will know what systems exist on their network and have security controls to mitigate new vulnerabilities.

3.2. Phase 2 – Requirements and Preparation

When an organization has finished preparing for penetration testing and application security testing, the testing team must first select a methodology and determine the goal of the testing. As is previously noted, the BSIMM is one of the most prescriptive approaches to software security. The BSIMM program provides 112 practices for secure software development. Several procedures for exist for penetration testing. The PTES contains hundreds of specific steps that a tester can take when executing a penetration test of any size. The OWASP testing guide is a suitable choice, but only provides testing steps for web-based applications. For providing quantitative measures of risk, the testing team may consider using the Open Source Security Testing Methodology Manual (OSSTMM). The OSSTMM provides specific ways to measure risk within a system and broad statements about what a tester should check (Merkow & Raghavan, 2010). It competes neither with the PTES or the OWASP testing guides since the OSSTMM does not supply specific technical checks to perform on a system (OWASP, 2016). Regardless of the methodologies and activities an organization select, it is important to select a process and communicate it to project managers, business owners, and IT owners.

3.3. Phase 3 – Prioritizing Systems

The third phase uses both qualitative and quantitative measures to build an application risk rating. First, either the security testing team or a separate governance, risk, and compliance (GRC) team should conduct a qualitative risk assessment. At a minimum, this assessment should capture the exposure value and data classification of the system. The technical security assessment team then executes the report card scoring process to produce a quantitative score. Qualitative scores should then be assigned a numeric value, with more risky systems having a lower rating. Figure 1 shows a potential rating scheme. These values should then be multiplied together to obtain a priority value for each system. After performing this process on several systems, then the full tests can begin on this prioritized list. When the organization learns of a new application, all phase 3 activities should be conducted and added to the queue for testing.

Risk Rating	Quantitative Multiplier
Critical	0.75
High	1.00
Moderate	1.25
Low	1.50
Very Low	1.75

Figure 1. Sample numeric ratings for qualitative risk levels

3.4. Phase 4 –Security Assessments and Treatment

During the final phase, technical security testers conduct an assessment of a specific system or application using the methodology defined in phase 2. This process may cover any number of security activities including threat modeling, Static Application Security Testing (SAST), Dynamic Application Scanning Tools (DAST), penetration testing, and other measures. After completing a test, it is essential to track any findings for remediation. As part of the critical controls implementation, the organization should already have a process in place for remediating vulnerabilities or handling the residual risk.

4. Web Application Tests

4.1. Administratively Observable Tests

The choice of programming language for a web application can significantly affect the overall security posture of the application. Some languages, such as ASP.NET and J2EE, have a great deal of security-related features built into the language. For example, the ASP.NET framework provides robust protection against reflected cross-site scripting and ViewState tampering. ASP.NET also provides features such as cross-site request forgery (CSRF) protection, SQL injection defense with Linq, and configuration file encryption. While it is possible to configure applications written in PHP in a secure manner, this requires significant effort (Meier et al., 2006). It is also important that frameworks, such as .NET or Java, use supported versions. For this test, languages such

as J2EE and ASP.NET with supported frameworks receive full points while Ruby, Perl, ASP.NET 1.0, and PHP receive no points.

Web applications should run under a service account with minimal permissions. The application should only require read access to most web-accessible folders. While it is possible for an application to write files securely to web-accessible folders, the potential for abuse is much greater than if this capability is never coded. Web application and database service accounts should never be administrators. Standard operating system tools can retrieve this information. Figure 2 demonstrates using netstat and ps to retrieve a process owner in Linux for a service listening on port 80 and Figure 3 illustrates the use of the Internet Information Services (IIS) management console to find this information. Analysts should award full points only if an application has read-only access to web-accessible folders. Partial credit is possible if the application uses write access judiciously. A small amount of partial credit is possible if an application has read/write access to the web directories. If an application runs with administrative permissions, then no points are awarded.

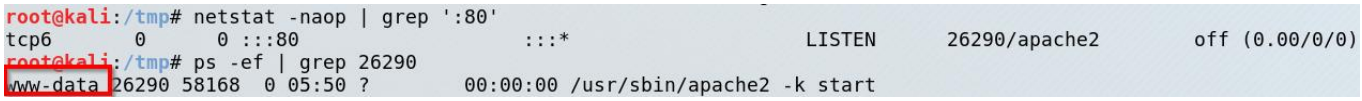

Figure 2. Locating the process owner from a port number on Linux.

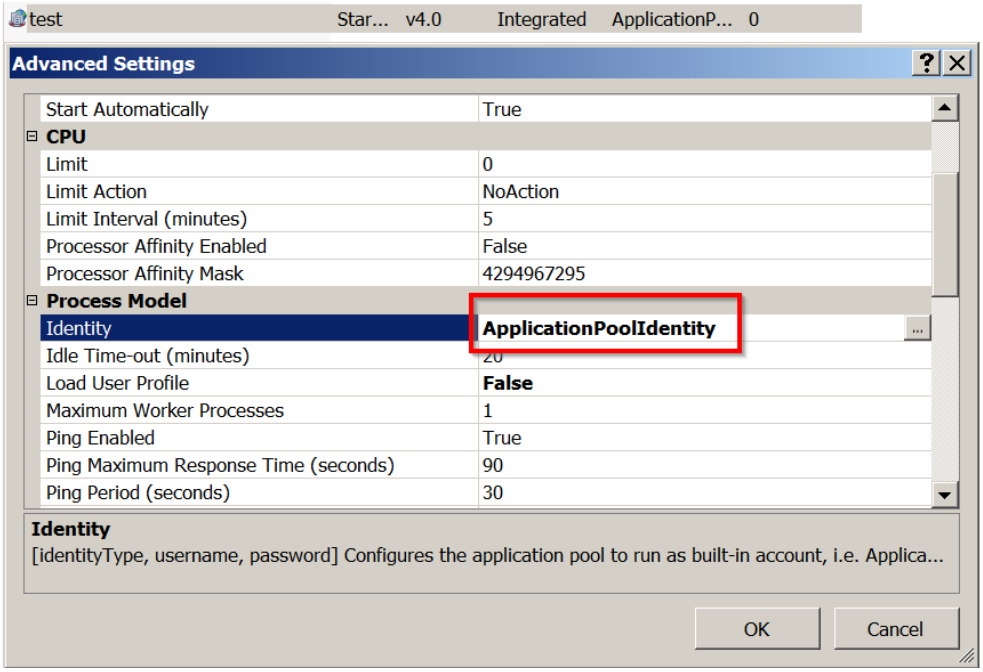

Figure 3. Viewing the application pool identity in Internet Information Services.

Often, web applications utilize standard web servers such as Microsoft's Internet Information Services, the Microsoft HTTP API, JBoss, Apache, or other similarly tried and tested components. However, it is possible that developers will rewrite standard web server functionality or use a solution that does not receive the same level of scrutiny as more popular server platforms. In these scenarios, analysts must also test core server components as vigorously as the web application itself. If administrative access is unavailable, tools such as Nikto are often sufficient to fingerprint a web server. However, it is more reliable to view the installed software on the server to determine application versions. Analysts should check both web server versions (such as IIS or Apache) as well as framework versions (such as PHP and the .NET framework). Figure 4 depicts using the apachectl command to view a server's version of Apache. Applications that use a well-tested server technology would receive full points. The analyst may award partial points for publicly available server technology that has not received the same level of scrutiny as the previously mentioned web servers. Custom implementations receive zero points.

Figure 4. Using apachectl to check a server's apache version.

Although attacks such as SQL injection leverage an application's database to affect a system's security, the use of a database still provides a measure of security versus other forms of data storage. Modern databases support authentication, encryption, and disaster recovery. In contrast, if an application chooses to leverage a textual data storage mechanism, such as eXtensible Markup Language (XML), or a proprietary storage mechanism then it is unlikely that all the security features offered by a full database will be present. SQL injection attacks against such a system may not be feasible, but other attack categories such as XPATH injection are available to malicious actors. The most reliable way to check the data storage mechanism is to interview system owners and then verify by checking software versions on the test systems. Use of a database such as Oracle, Postgres, MySQL, or Microsoft SQL Server will receive full points. SQLite databases, XML, and flat file data storage receive zero points. Systems that serve static content and have no form of back-end data storage will still receive full points.

Application Programming Interfaces (APIs) provide a mechanism for programmatic access to an application's functionality. Client-side mobile or thick client applications often consume these APIs. A secure API is certainly possible, but testing shows that direct access to an API often results in security vulnerabilities. Languages such as ASP.NET provide controls such as the RegularExpressionValidator control that are very effective at preventing injection attacks. In an API, the developers must manually implement these protections on the server. However, this is a simple step to forget, and testers must specifically configure DAST to test these APIs. Some indicators of the presence of an API are Simple Object Access Protocol (SOAP) or JavaScript Object Notation (JSON) requests or common web service extensions, such as svc or asmx. If an application does not contain an API, then full points are awarded. The presence of an API or web service results in zero points. Use of a framework such as the Model-View-Controller (MVC) framework does not count as an API for scoring.

Strong authentication is a requirement for any application that processes sensitive information. However, not all authentication mechanisms are equal. Creating custom authentication mechanisms can cause many security issues. OWASP scores broken authentication as the number two vulnerability in its top 10 (OWASP, 2013). Instead, applications should leverage common authentication frameworks and single sign-on technologies whenever possible. Figure 5 shows an example of an IIS server configured to require a specific Windows group access to a website. Analysts should award full points for the use of common single sign-on technologies such as Google Authenticator, open token, certificate-based authentication, Active Directory, or similar solutions. Analysts should award no points if an application uses a custom authentication method.

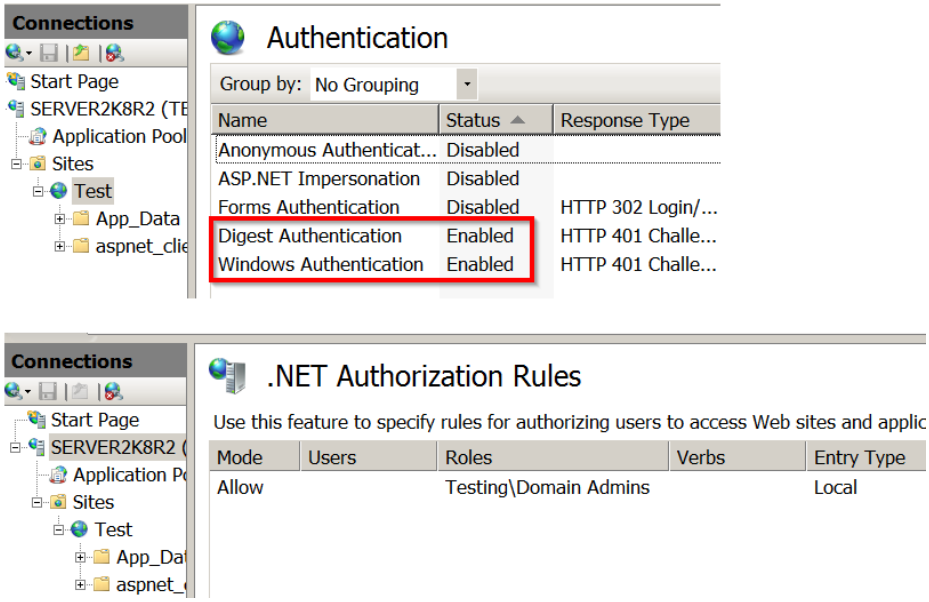

Figure 5. IIS configured to allow only a specific group website access.

4.2. Dynamic Tests

When an application uses passwords for authentication, it is often necessary to provide password reset functionality. Issues such as security question choice, storage of security questions, and brute force attacks will often arise in these implementations. This functionality is usually in the form of a "Forgot Password" link on a web site's login page. If applications reset passwords using out-of-band methods, then an application should receive full points. Some valid out-of-band methods are SMS and manual

verification by a customer service representative. Using other methods such as secret question/secret answer receive no points.

If the application requires passwords, the complexity requirements influence how easy it will be to acquire control of an account using brute force techniques. Emerging government standards recommend a minimum length of eight characters and a maximum length of at least sixty-four characters. The list of acceptable characters should be all printable characters, including UNICODE characters. Applications should also check passwords against a blacklist of trivial and recently compromised passwords (Grassi et al., 2015). Finally, the use of multi-factor authentication is also necessary to receive maximum credit for this test. Applications that implement only some of the defenses against weak passwords can obtain partial credit.

File uploads in web applications are another significant source of vulnerabilities. Like APIs, a developer can securely provide the ability to upload files to a web server, but this requires several checks (Ullrich, 2009). While discerning if these security controls are present in an application is difficult, finding whether or not file upload functionality exists is usually simple. Testers should interview system owners to determine if an application supports file uploads and conduct a cursory review to discover this capability. DAST tools such as Burp proxy will alert an analyst to this functionality, as Figure 6 shows. If an application allows file uploads then zero points should be awarded; otherwise, the application should receive full credit.

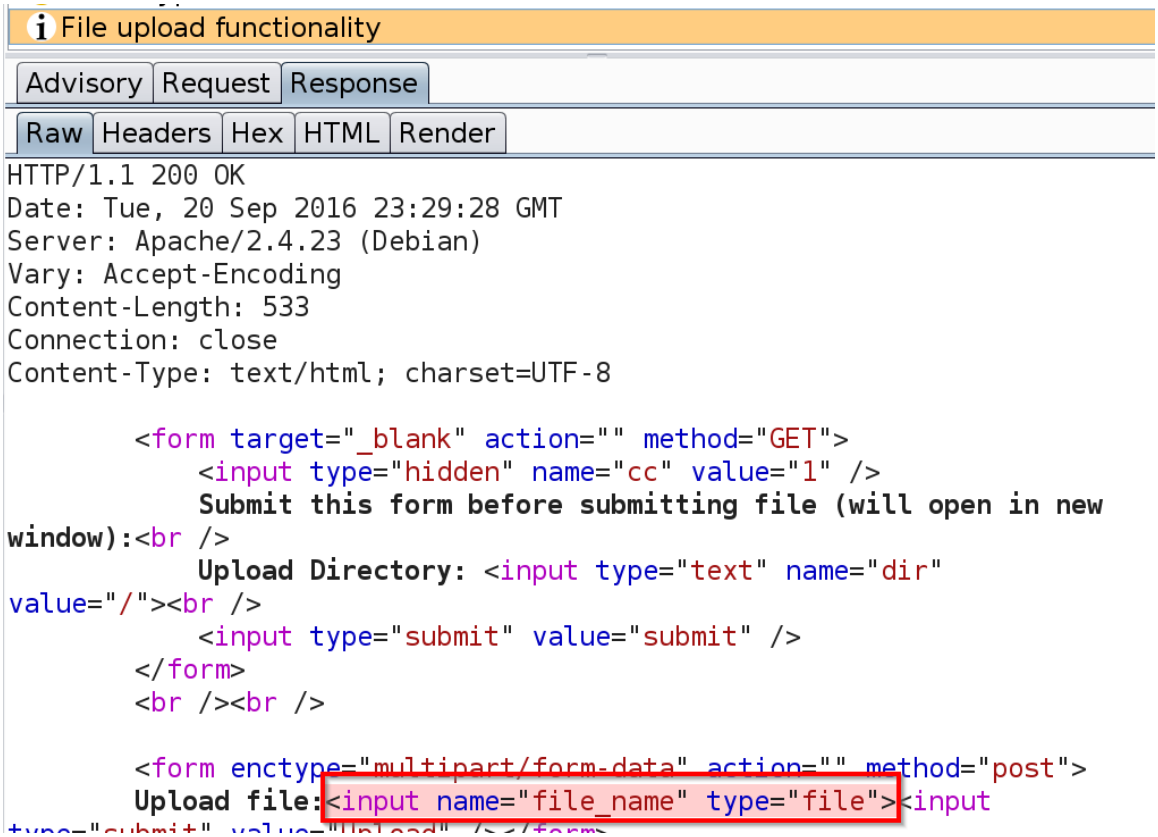

Figure 6. Burp proxy detecting a file input control.

Cross-site request forgery (CSRF) vulnerabilities arise due to lack of entropy in web requests. The client must submit a random value as part of each request to the server to prevent this category of attack. Fortunately, analysts can easily observe this countermeasure during an initial assessment. Often this will manifest as a random token in a hidden form field or the ASP.NET ViewState. Apache Tomcat Manager embeds CSRF tokens in each request, even those using the GET methods. Figure 7 shows an example of the Apachhe CSRF token. Figure 8 displays the ASP.NET MVC CSRF token, which manifests as a hidden form field called "__RequestVerificationToken (Wasson, 2012)." It is also possible to implement CSRF defenses by checking the "Origin" or "Referer" headers (Wichers, Petefish, & Sheridan, 2016). DAST tools such as Portswigger's Burp or OWASP's Zed Attack Proxy (ZAP) can be used to verify that an application uses one of these techniques by scanning a single POST request (Portswigger, n.d.). As with all checks in the report card, the purpose is not to find all vulnerabilities, just to determine if developers have made a reasonable attempt to implement security

controls. If testers find evidence of CSRF prevention techniques, then the application should receive full points.

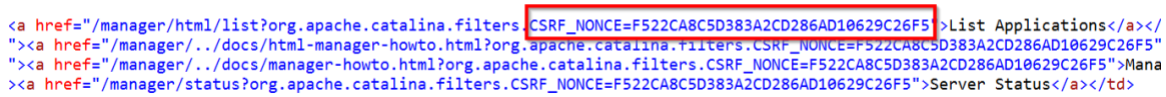

Figure 7. Apache Tomcat Manager with CSRF protection on each link.

```
<form action="/Home/Test" method="post">
    <input name="__RequestVerificationToken" type="hidden"
           value="6fGBtLZmVBZ59oUad1Fr33BuPxANKY9q3Srr5y[...]" />
    <input type="submit" value="Submit" />
</form>
```

Figure 8. CSRF protection in an ASP.NET MVC application (Wasson, 2012).

Input sanitization is a concept that is central to preventing many different types of attacks. For both security and user experience, regular expressions should be used to ensure that user input is in the format expected by the application. Analysts can quickly check ASP.NET applications for the presence of the various "Validator" controls, such as RegularExpressionValidators and RequiredFieldValidators. These controls automatically run the expected input validation checks on both the client and server. Other languages are not quite so obvious and require manual verification. Analysts should look for fields such as email address, dates, and social security numbers, then submit malformed data using an intercepting proxy. The expected result is that the application should produce an error message that the input is improperly formatted.

Cross-site scripting (XSS) and HyperText Markup Language (HTML) injection attacks result when a server directly sends uncontrolled data to a user's web browser without proper encoding. Analysts can observe whether or not an application appropriately encodes data by supplying potentially dangerous characters to input fields. If the application either reflects this input or places it into an unsafe control, such as a .NET label, analysts should observe if the dangerous characters are encoded. A man in the middle proxy should be used for this task, as modern web browsers frequently automatically encode characters to prevent XSS attacks. Figure 9 shows an HTML-encoded string. An exhaustive test of the application is not necessary. Analysts only

should check a sample of locations where their input is reflected. If any failures to encode in a dangerous control are detected, then the analyst should award zero points.

```
HTTP/1.1 200 OK
Date: Sun, 25 Sep 2016 20:02:19 GMT
Server: Apache/2.4.23 (Debian)
Content-Length: 32
Connection: close
Content-Type: text/html; charset=UTF-8

Dangerous output:&lt;'&quot;&gt;
```

Figure 9. HTML-encoded characters in an HTTP response.

Attackers often target a web application's cookies for theft since they commonly allow the attacker to act as a legitimate user or contain other sensitive data. This statement is especially true for session tokens that are regularly used to track a user's login state. These values are often disclosed either by including cookies in insecure transactions or through a cross-site scripting vulnerability within an application. Fortunately, modern web browsers support two flags that will protect these cookies under many circumstances, and analysts can easily observe the flags. Analysts can simply use the application and then check the cookies using a tool such as FireBug to ensure that the application uses these flags. Figure 10 demonstrates using FireBug to check the status of the HTTPOnly and Secure flags. Analysts should award points based on the percentage of cookies using each of the flags. Applications that do not use cookies should receive full points.

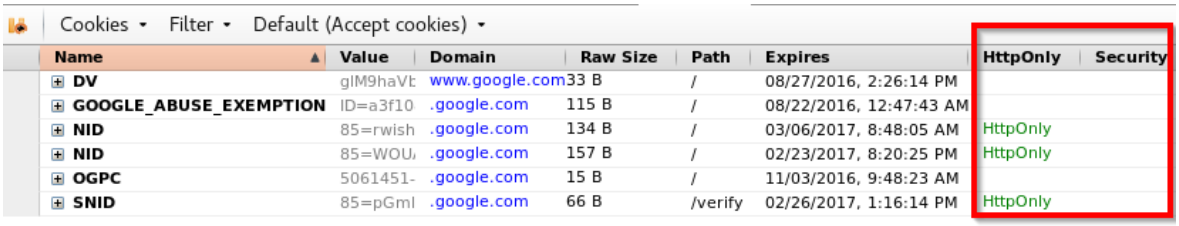

Figure 10. Using FireBug to view cookie flags

Most applications only utilize a small subset of the full list of HTTP request methods. Even so, applications often support far more of these request methods than are necessary for a system to function. Even if the application does use the extra methods,

they often indicate that the application is more complex than standard web applications and is, therefore, worthy of extra scrutiny. Nikto is one of the simplest tools to confirm which HTTP request methods a server supports. Using Nikto to check HTTP request methods is shown in Figure 11. If a server supports only a minimal set of methods – GET, POST, OPTIONS, and HEAD – then it should receive full points. Analysts can then award partial credit if the server supports limited additional methods.

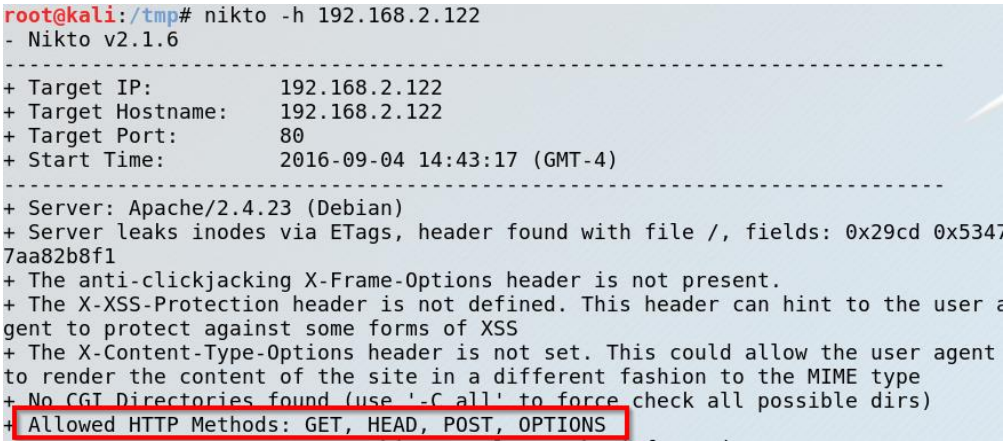

Figure 11. Using Nikto to check allowed HTTP request methods

The current threat landscape dictates transport layer encryption for all applications, including those that are not accessible from the public internet. Without encryption, attackers are free to monitor or modify sensitive communications. At a minimum, when the client and server send sensitive information, this must operate over an encrypted channel. Analysts can ensure that any pages with such sensitive information use SSL by looking at the uniform resource indicator (URI) in a web browser or an intercepting proxy. An application can achieve full points if it sends all credentials, credit card data, personally identifiable information (PII), protected health information, or other sensitive data over an encrypted channel. Analysts should use their discretion to award partial credit.

Misconfigured error pages often disclose details of an application's internal workings. The best practice is that applications should display only a generic message that an error occurred and possible reference numbers (Keary, 2007). Analysts can send malformed requests and requests containing malicious content to a server and observe the errors returned. Figure 12 displays a 404 response page disclosing the web server's

version information. If the error message contains only information such as the HTTP response code and contact numbers, then the application receives full credit.

Figure 12. A default error page disclosing information about the server

Websites have the ability to define limited violations of the same origin policy (SOP). The SOP dictates that one site cannot access the resources of another. Enterprise applications must often allow such resource sharing to allow interoperability between various sites. These policies are often improperly implemented, especially in commercial applications, increasing the likelihood of CSRF vulnerabilities. An analyst can check these policies through the use of the OPTIONS HTTP method, crossdomain.xml, and clientaccesspolicy.xml. Simple tools, such as Nikto, perform these checks automatically. Using Nikto to perform this check is illustrated in Figure 13. If an application does not allow any cross-origin resource sharing or limits the scope to specific sites, then analysts should award full points. Partial credit is possible if the sharing scope is a single domain, such as companyname.com.

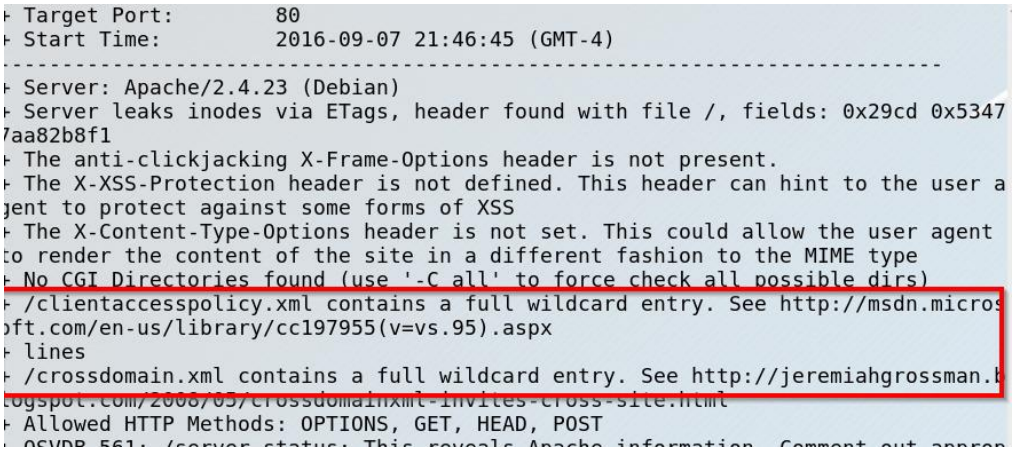

Figure 13. Using Nikto to check cross domain policies for Flash and Silverlight

5. Relation to the Critical Security Controls

Report cards are not intended to implement any of the twenty critical control families but can enforce some of the Application Security sub-controls. Analysts verify vendor or development support of both applications and frameworks, which verifies adherence to CSC 18.1. The report card process also performs limited verification that an application checks both input and output validity, which is required by CSC 18.3. However, it is unreasonable to expect a comprehensive result from such an abbreviated process. Organizations should utilize SAST and DAST tools to ensure that applications properly handle all system inputs and outputs. Finally, analysts satisfy CSC 18.5 by checking that an application's error messages do not disclose sensitive information to the end-user. If organizations expect additional requirements – such as a web application firewall, separate production and development environments, or secure software development training – then organizations may also include these as checks in the report card process.

6. Future Research

Security testing activities are time-intensive and may carry risk to system availability. Several efforts to deliver faster, more complete, and safer tests are underway. Advances in these systems may drastically alter or perhaps obviate the need for conducting preliminary technical investigations such as those in this paper.

The Defense Advanced Research Projects Agency (DARPA) funded several efforts to automate the security assessment process. In 2015, DARPA funded the Cyber Independent Testing Laboratory (CITL) to devise a system to rate the security of applications. When this project is complete, it will use automation to create a consumer report security score for many applications (Simonite, 2016). The markers observed in this process are all technically observable criteria and manual checks may not be necessary after projects such as CITL's come to fruition. DARPA also sponsored the 2016 Cyber Grand Challenge, an all-machine hacking tournament. Competitors in this challenge created applications that can discover and patch security vulnerabilities in applications (DARPA, 2016). As this concept expands, both the need and focus of penetration testing and application security efforts are likely to change.

There are several ways to improve the report card system. First, the report cards would benefit from a greater breadth of experience. With only a single tester supplying input to the process, both the tests and point values for tests are likely biased. A greater breadth of applications would also improve the report card's accuracy. All applications tested were used in a single financial services company. Other types and sizes of businesses would serve to reduce bias. Analysis tools and scripts could automate many of the tests in the report card system. Automation would greatly reduce the time and skill requirement to implement this process. Finally, there are many application types other than web applications and mobile applications. Separate report cards for thick client and web APIs would allow the system to apply to a much larger percentage of enterprise applications.

7. Conclusion

Application security and penetration testing are both part of the critical security controls. However, these practices come near the end of a long line of security controls. When it is time to undertake these tests, organizations have a vast number of systems in operation, which will require a huge amount of time to test. Standard qualitative risk assessment measures rarely have sufficient data points to stratify all these systems so that testing can focus on the systems that pose the greatest risk to the organization. By imposing a larger battery of technical tests with quantifiable scores and combining them

with information about system value, the organizational risk can be better quantified. This abbreviated examination allows a company to focus their exhaustive and resource-intensive testing activities where they can provide the greatest benefit.

The goal of the report card tests is to indicate how likely a software application is to present a security risk without requiring a large time investment. Tests must not only be useful for security, but analysts must be able to perform all the checks in a short period. These factors greatly limit the number and type of tests in the report cards. Often, security questions are difficult to answer in an absolute sense, involving checks on a large number of system inputs or requiring a great deal of specialized training. The tests selected for the report cards are quick, repeatable, and relatively simple.

These prioritization techniques are intended to be just a part of the enterprise security program. If resources such as time, budget, and personnel were never an issue, then there would be no need for such activities. Real-world organizations must make difficult choices about where to focus attention. By using the tests outlined in the report cards, analysts can provide clarification for these choices across a wide range of systems in a short time.

References

Ard3n7. (2013). *Introduction to Application Risk Rating & Assessment*. Retrieved September 10, 2016, from http://resources.infosecinstitute.com/introduction-to-application-risk-rating-assessment/.

Bing, C. (2016). *NSA: no zero days were used in any high profile breaches in the last 24 months*. Retrieved September 20, 2016, from http://fedscoop.com/nsa-no-zero-days-were-used-in-any-high-profile-breaches-over-last-24-months.

Center for Internet Security. (2016). *The CIS Critical Security Controls for Effective Cyber Defense (v6.1)*. Retrieved September 16, 2016, from https://www.cisecurity.org/critical-controls/documents/CSC-MASTER-VER61-FINAL.pdf.

Conrad, E., Misenar, S., Feldman, J., Riggins, K. (2010). *CISSP Study Guide*. Burlington, MA: Syngress.

DARPA. (2016). The Cyber Grand Challenge. Retrieved September 10, 2016, from https://www.cybergrandchallenge.com.

Grassi, P., Fenton, J., Newton, E., Perlner, R., Regenscheid, A., Burr, W., Richer, J., Lefkovitz, N., Danker, J., Choong, Y., Greene, K., Theofanos, M. (2015). *Draft NIST Special Publication 800-63B Digital Authentication Guideline*. Retrieved September 25, 2016, from https://pages.nist.gov/800-63-3/sp800-63b.html.

Keary, E. (2007). *Error Handling*. Retrieved September 10, 2016, from https://www.owasp.org/index.php?title=Error_Handling&setlang=en.

McGraw, G. (2006). *Software Security Building Security In*. Upper Saddle River, NJ: Pearson Education Inc.

Merkow, M. & Raghavan, L. (2010). *Secure and Resilient Software Development*. Boca Raton, FL: CRC Press.

Meier, J. Mackman, A. Dunner, M. Vasireddy, S. Escamilla, R. & Murukan, A. (2006). *Building Secure ASP.NET Pages and Controls*. Retrieved September 10, 2016, from https://msdn.microsoft.com/en-us/library/ff648635.aspx.

Meucci, M., Mueller, A. (2014). *OWASP Testing Guide v4*. Retrieved September 13, 2016, from https://www.owasp.org/images/1/19/OTGv4.pdf.

NIST. (2012). *NIST Special Publication 800-30 Guide for Conducting Risk Assessments.* Retrieved September 25, 2016, from http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-30r1.pdf.

NIST. (2013). *NIST Special Publication 800-53 Security and Privacy Controls for Federal Information Systems. Retrieved September 25, 2016, from http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf.*

Northcutt, S., Shenk, J., Shackleford, D., Rosenberg, T., Siles, R. & Mancini, S. (2006). *Penetration Testing: Assessing Your Overall Security Before Attackers Do*. Retrieved September 7, 2016, from https://www.sans.org/reading-room/whitepapers/analyst/penetration-testing-assessing-security-attackers-34635.

OWASP. (2013). *Top 10 2013*. Retrieved September 10, 2016, from https://www.owasp.org/index.php/Top_10_2013-Top_10.

OWASP. (2016). *Penetration Testing Methodologies*. Retrieved September 24, 2016, from https://www.owasp.org/index.php/Penetration_testing_methodologies.

Peake, C. (2003). *Red Teaming: The Art of Ethical Hacking*. Retrieved September 8, 2016, from https://www.sans.org/reading-room/whitepapers/auditing/red-teaming-art-ethical-hacking-1272.

Portswigger. (n.d.). *Using Burp to Test for Cross-site Request Forgery*. Retrieved September 10, 2016, from https://support.portswigger.net/customer/portal/articles/1965674-using-burp-to-test-for-cross-site-request-forgery-csrf-.

Scarfone, K., Souppaya, M., Cody, A., & Orebaugh, A. (2008). *NIST Special Publication 800-115 Technical Guide to Information Security Testing and Assessment*. Retrieved September 7, 2016, from http://dx.doi.org/10.6028/NIST.SP.800-115.

Simonite, T. (2016). *How Public Shame Might Force a Revolution in Computer Security*. Retrieved September 10, 2016, from https://www.technologyreview.com/s/602104/how-public-shame-might-force-a-revolution-in-computer-security/.

Ullrich, J. (2009). *8 Basic Rules to Implement Secure File Uploads*. Retrieved September 10, 2016, from http://software-security.sans.org/blog/2009/12/28/8-basic-rules-to-implement-secure-file-uploads.

Wasson, M. (2012). *Preventing Cross-Site Request Forgery (CSRF) Attacks in ASP.NET Web API*. Retrieved September 21, 2016, from http://www.asp.net/web-api/overview/security/preventing-cross-site-request-forgery-csrf-attacks.

Wichers, D., Petefish, P., & Sheridan, E. (2016). *CSRF Prevention Cheat Sheet*. Retrieved September 10, 2016, from https://www.owasp.org/index.php/CSRF_Prevention_Cheat_Sheet.