Star East Paper

Application Security: What Testers Can Do

Prepared for:	Star East 2015
	

Tessolve Semiconductors Private Limited

Electronic City Phase II,

Bangalore – 560 100,

Karnataka, India.

Tel: +91 80 4181 2626,

+91 80 6816 2626

e: sales@tessolve.com

www.tessolve.com

TESSOLVE
A Hero Electronix Venture

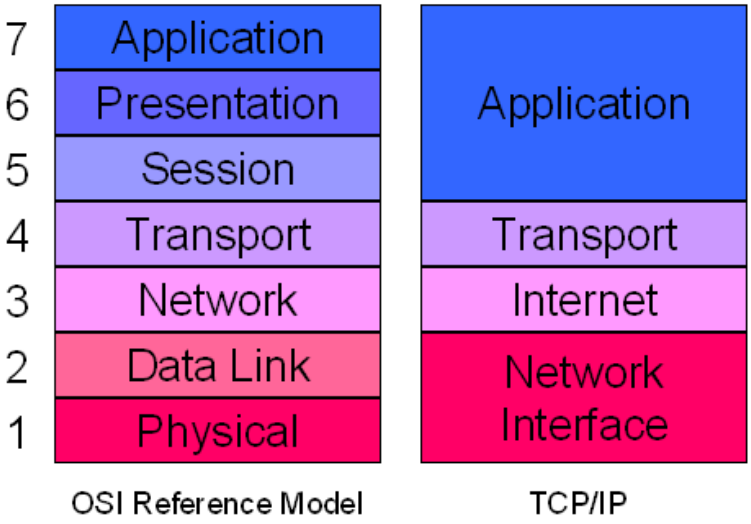
Non-Functional. Adjective: *1) Not having any particular purpose or function. 2) Not operating or in working order.* (Oxford English Dictionary)

Hmm, that definition seems to differ substantially from the accepted IT definitions of 'non-functional' which are best summarized as: 'All the stuff that's difficult to do or measure, and best left for someone else instead of me'. So long as we can pigeon-hole security as something outside the mainstream project activity it is easy to forget about it or transfer responsibility to outsiders.

And yet. What if for one day we considered application security was something other than non-functional? Perhaps it is *extra-functional*, or just plain functional? There is enough evidence for us to see security breaches are currently a massive problem. Take *Marathon Oil* for example. Marathon invested millions of dollars surveying the ocean floor for potential oil deposits, then submitted their bids to the Indonesian government for the undersea fields they wanted to drill. For every single field, a Chinese oil company outbid Marathon by the smallest possible margin, without bidding for any other fields or ever being seen undertaking surveys. The loss of potential earnings probably runs into billions of dollars.

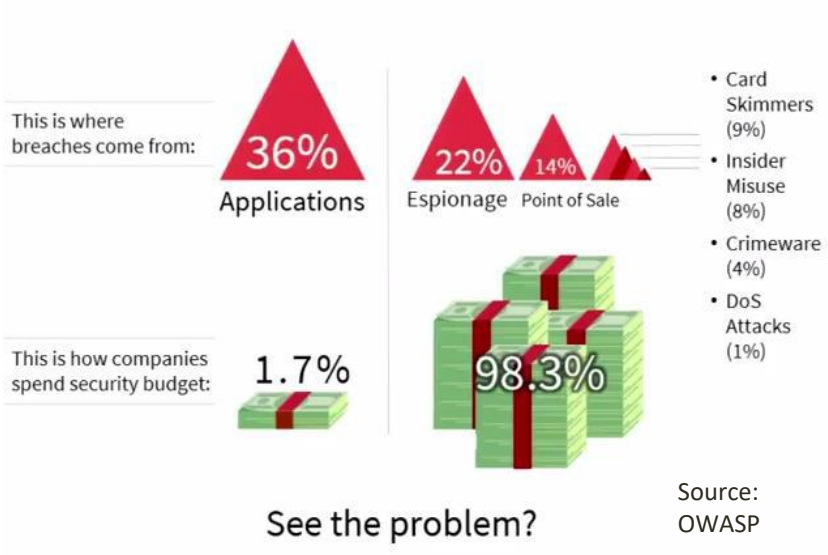
Sure enough, when experts looked closely at Marathon's IT system it was found to have been breached and we can only assume from the circumstantial evidence there was a connection between the hack and the bidding events. Elsewhere mega-breaches have become commonplace and I have even heard CISSP certified security professionals advise it is impossible to prevent breaches so the focus should be on recovery after the breach. That argument would not make me feel good if ISIS took control of the US nuclear arsenal and the President assured us the recovery process would re-stock the silos with replacement missiles next week.

Why is hacking so successful despite enterprises spending $46 billion dollars on cyber-security last year? One very important concept to consider is the difference between the TCP/IP transport layer and the application layer.



According to the Open Web Application Security Project (OWASP), only 1.7% of security budgets are spent on defending the application layer, despite most attacks targeting the applications and most breaches occurring at the application level.

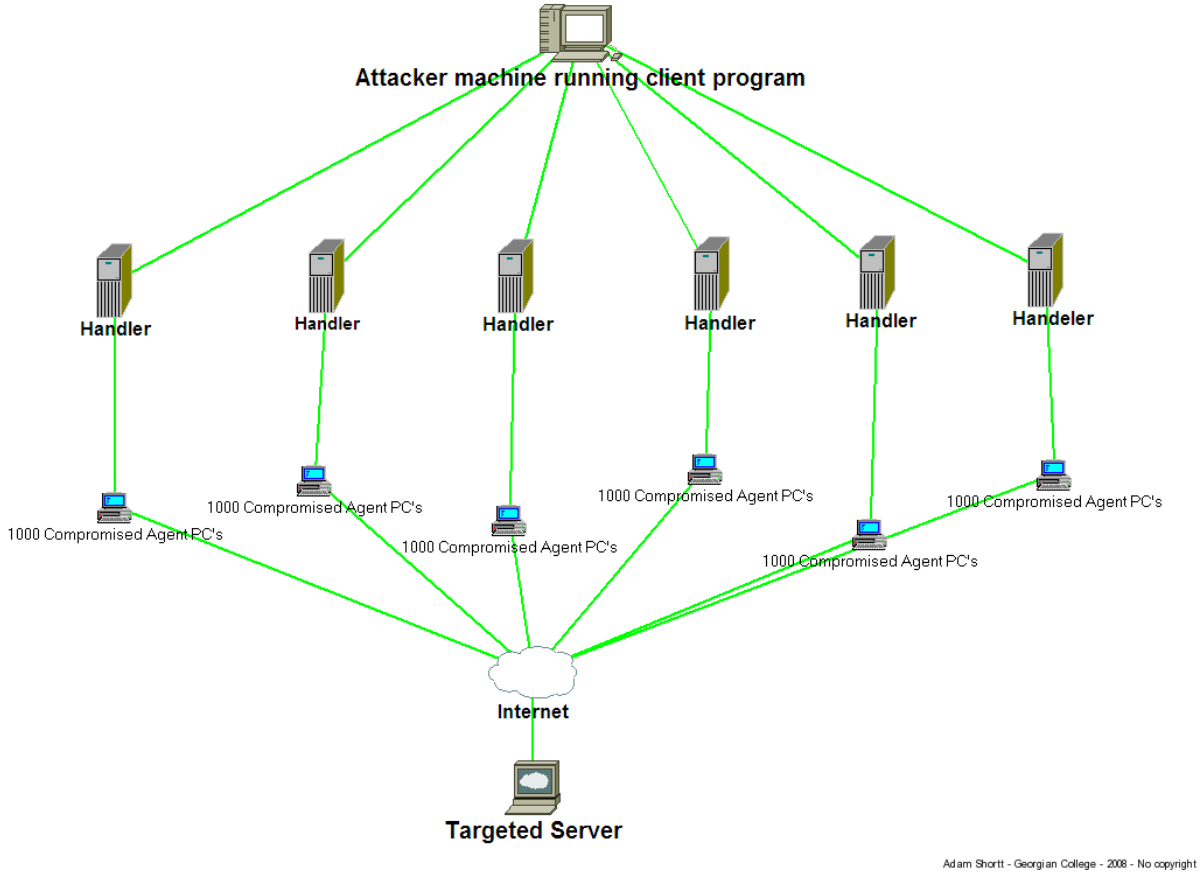
Other research suggests that perhaps the application security budget may be around 15%. Certainly, it is far smaller than the established network / perimeter defence budget which is well stocked with expensive yet significantly ineffective tools such as firewalls, Intrusion Prevention Systems (IPS), Intrusion Detection Systems (IDS) and anti-virus software. These defences rely primarily on pattern-matching messages travelling through the network with their catalogues of known-bad scripts. At least they do that if the message is in clear Hyper Text Transfer Protocol (HTTP), but no valuable data should be transported unencrypted, hence the 25% and growing quantity of traffic that layers HTTP over the Transport Layer Security (TLS) cryptographic protocol to create HTTPS. In case you were wondering, TLS version 1.0 replaced Secure Sockets Layer (SSL) after SSL version 3.0.



Encrypting network traffic by using HTTPS prevents wiretapping and 'man-in-the-middle' attacks, but also prevents network defences from detecting an intrusion because they cannot read the inbound or outbound messages unless they are provided with the encryption keys and perform a benign 'man-in-the-middle' interception themselves. Allowing an IPS to break encryption and log events, however well intended, presents new opportunities for attackers and new risks for defenders. Examining encrypted messages in-flight also creates performance issues. NSS Labs report an 88% reduction in transactions per second when HTTPS is decrypted, and the greater the flow rate the lower the percentage sampled in-line becomes. Unstructured internet data is growing at around 60% per year and will probably increase from 7.9 zettabytes this year to 82 zettabytes by 2020.

Big volume businesses like Yahoo already use 10 Gigabit per second (Gbps) Ethernet and expect to have standard 100 Gbps network connections by 2020. That increase in flow rate will reduce the real-time security window from 67 nanoseconds to 6.7 nanoseconds per frame. In other works the network defences will have to load a message into a frame, decide if it is malicious, then either block it or allow it onwards in the time it takes a photon of light to travel two metres. A top-end next generation firewall such as the Palo Alto 7050 can cope with 120 Gbps, but the cheapest model starts at $300,000 and each one consumes 2.4 Kilowatt of power. This kind of defence is becoming unaffordable and looks like attempting to make propeller-driven airplanes fly at supersonic speeds. No matter how much money and effort is input, the diminishing return on investment will defeat the buyer before the objective can be achieved.

Meanwhile attackers are taking advantage of the asymmetric economics of cyber-security. For as little as $20 a malicious script can be purchased on the dark-web (a marketplace for illegal activity) then hijack thousands of unsuspecting computers to attack the multi-million dollar network defences of a target, and possibly defeat them.



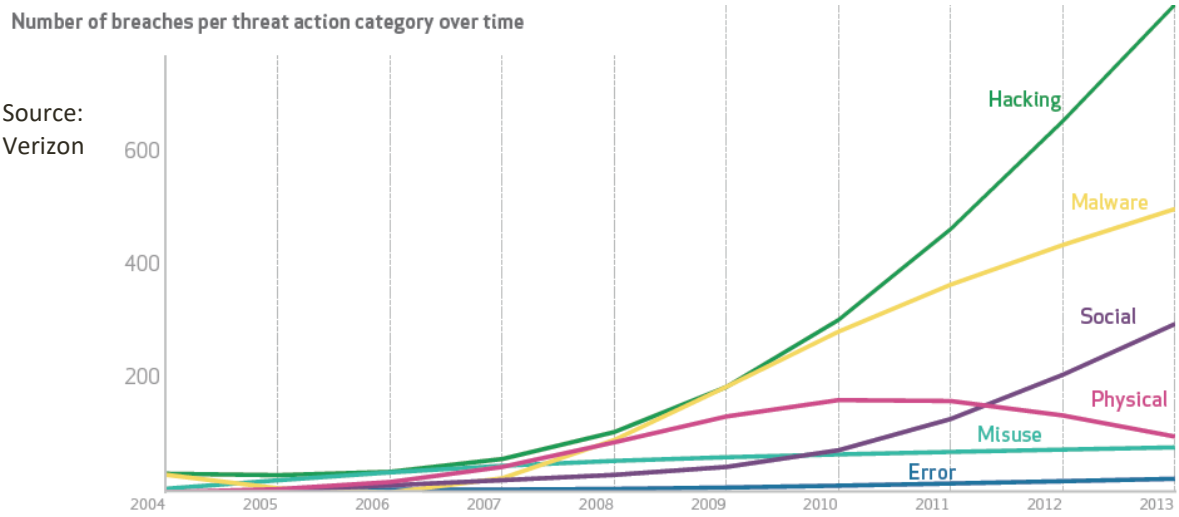
What about all the globally agreed security standards your organization follows? Surely passing an ISO 27001 Information Security Management System (ISMS) audit proves your IT systems are secure? Unfortunately, standards set a floor not a ceiling for security, and many organizations have still suffered breaches after passing their security audits. One weakness of ISMS audits is that they only verify the defined processes are correctly implemented.

If there are no defined processes for developers to follow secure coding standards, or testers to perform thorough security testing, the audit will simply not consider them.

The proof that our current security paradigm is not working can be seen in the Verizon research graph shown below.

Although an ISMS may contain insider misuse through defined disciplinary procedures, and limit breaches after physical loss of laptops by mandating password and encryption policies, the growth of breaches due to malware and hacking has now entered a golden era for criminals.

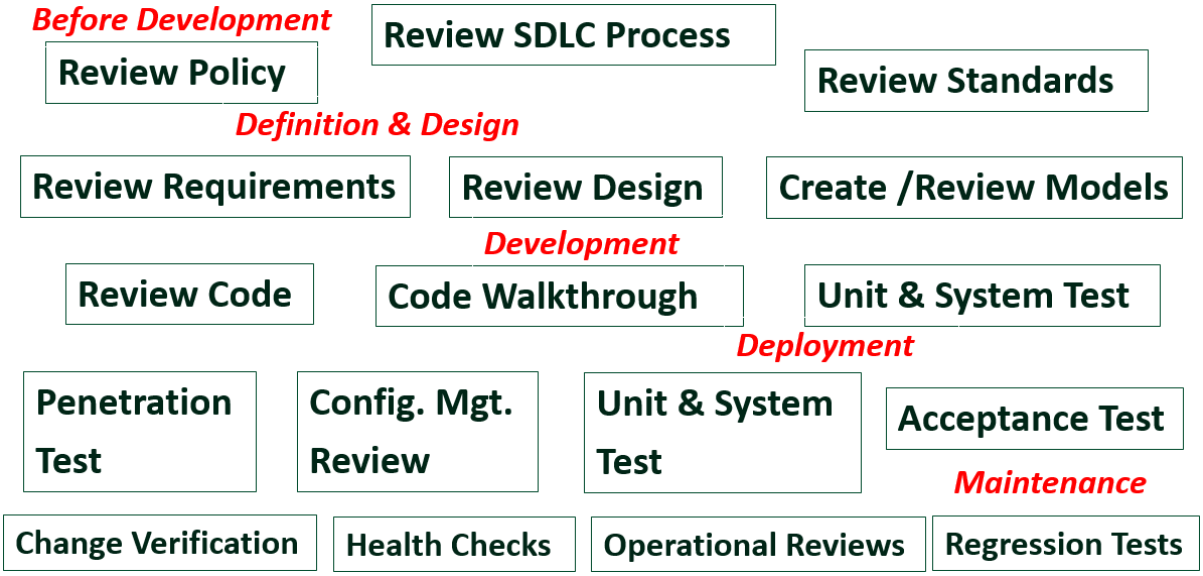
Source: Verizon



We need to reverse these trends, and work with limited or no budget to invert the asymmetric economics of cyber security in our favour. Perhaps there is no such thing as an un-hackable system, but if we can make it cost a hacker $10 to steal $9, they will give up attacking our systems. How can we start making software more secure, instead of buying more security software? Read on…

A first step is to see penetration testing and automated code reviews as just two parts of a much larger Secure Development Life Cycle (SDLC). Penetration testing is often used as a late tactical response to an ongoing strategic issue. If the project team have no understanding of application security, their relationship with penetration testers will be based not only upon trust but also ignorance, and that is a problem. Perceiving security as a black art permits its

aficionados to test against an undefined and incomplete mental criterion, instead of providing confidence that controls have been effectively implemented to manage prioritized security risks. Automated code reviews by generic tools are never 100% accurate because they are not created specifically for your system. Most issues are not generic, but deeply embedded in the bespoke business logic and application design. Automated code reviews generate false positives that waste investigation time, and more seriously leave false negatives that allow vulnerabilities to remain undetected. Just like penetration tests, automated code vulnerability scans are only a part of the bigger picture.



Using the model above, we can see the Secure Development Life Cycle starts with creating and regularly reviewing the whole process. A model best avoided is the 'build and fix' framework that reacts to problems as they arise. The benefit of getting a system into production will probably be outweighed by the long-term costs of damage control and trying to build quality in backwards. Other SDLC frameworks such as the *'V' model, Iterative*, and *Agile* approaches all have their usual strengths and weaknesses for building security into a project as they have for any other qualities. The important issue is to ensure security is fully included in the development process.

Before any development starts we can help senior management create or review the security policies (as appropriate) for acceptable use, risk management, vulnerability management, data protection, access control, business continuity, logging, auditing, personnel security, physical security, change control, email usage, incident response, and secure application development. The policies may be issue-specific (e.g. email policy) or system-specific (e.g. who has access to a database).

Standards are mandatory activities, actions, or rules that support the security policies. They could be data protection laws, IEEE standards, or company rules on data encryption in transit and at rest. At the next level of documentation there may be recommended guidelines and detailed security procedures. They might be dull, but they are useful.

At last we can look at the much more interesting area where half the security problems originate, the definition and design stage. All too often we test to verify customer requirements and use cases have been delivered, without considering misuse cases and what

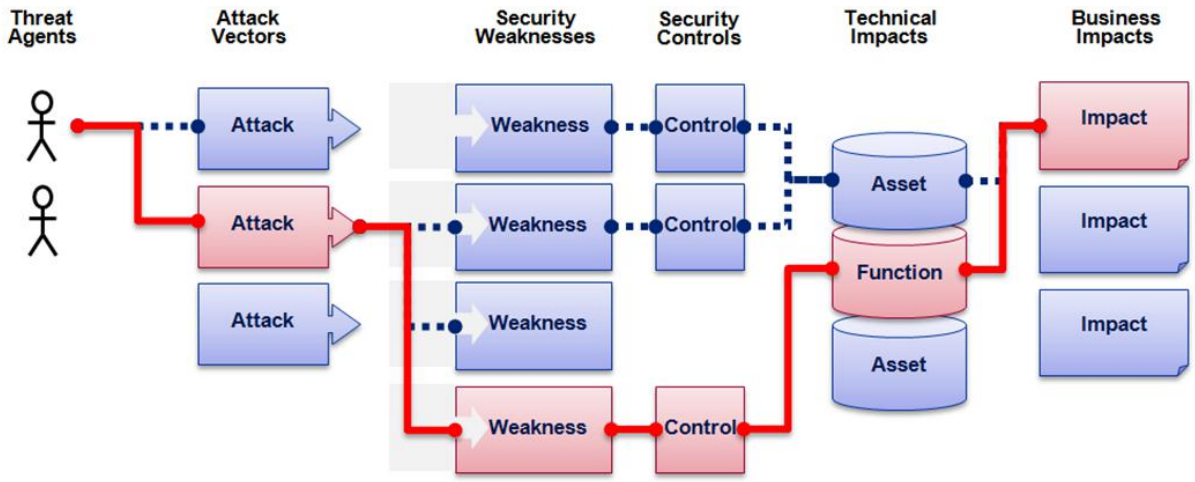
an attacker would try to do. The requirements review phase should identify the security requirements, the security risk assessment, privacy risk assessment, and risk-level acceptance. Accepting, avoiding, reducing, and transferring are all valid risk treatments. Ignoring risk is not.

The security requirements can be defined within the principle categories of **Confidentiality** (ensuring information is not disclosed to unauthorized parties), **Integrity** (ensuring data is accurate, complete, and protected from unauthorized modification), and **Availability** (ensuring data, systems, and resources are available to users in a timely manner). These requirements are sometimes known by the acronym C.I.A. Every security control addresses at least one of these three principles. Security requirements are derived from the applicable standards and regulations, plus the positive and negative application requirements.

The security testing objective is to validate the security requirements have been met by the effective application of security controls. Those controls are derived from threat modelling.

Threat modelling begins with decomposing the application by inspecting assets, functionality, and connectivity. The assets are then defined and classified into tangible (e.g. money) and intangible (e.g. reputation) and ranked by their importance to the business. The next step requires some expert guidance: Explore the potential vulnerabilities and threats to develop a realistic view of potential attack vectors. Once that is done, the mitigation strategy can be created by developing the mitigating security controls for every threat deemed realistic.

What if you don't have access to any expert guidance? Let us find a work-around. Consider this OWASP threat model.



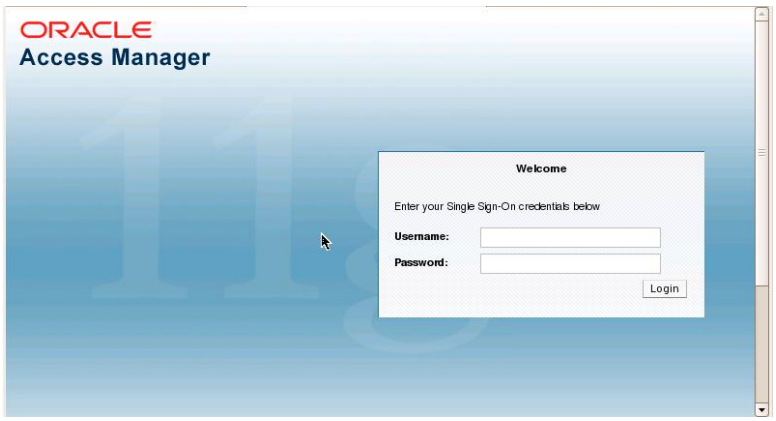
Threat Agents	Attack Vectors	Weakness Prevalence	Weakness Detectability	Technical Impacts	Business Impacts
App Specific	Easy	Widespread	Easy	Severe	App / Business Specific
	Average	Common	Average	Moderate	
	Difficult	Uncommon	Difficult	Minor	

On the left side we have threat agents. They might be financially motivated criminals, cyber-espionage spies wanting to steal secrets, politically motivated 'hacktivists', kids doing it for the lulz (fun), or insiders with a grudge. On the right side we have assets that need protecting. You already know more about those assets than any security expert brought into your organization from the outside world. You can imagine the business impact if the confidentiality, integrity, or availability of those assets were compromised. Now we need to fill in the part in the middle which requires some understanding of the subject. I have written application security testing procedures and development guidelines that condense a very large volume of material regarding the OWASP top-ten web application vulnerabilities into bite-sized chunks.

Unfortunately, despite what tool vendors may claim, there is no 'silver bullet' panacea for fixing application security vulnerabilities. If you can read and learn the contents of those documents, you will enter a small elite of testing professionals who have taken the time to understand the OWASP top-ten vulnerabilities. The systems your organization develops may well have different, or more than ten vulnerabilities, yet understanding these ten provides a great deal of leverage to understanding many other security issues. Absolutely nothing bad will come of trying to tackle them, as opposed to ignoring them.

The thought of spending time reading more documents might be frustrating, so let us push ahead now with what we have. We can imagine a system that needs protecting, and security controls (as detailed in the Tessolve documents) have been applied to prevent threats exploiting potential vulnerabilities. What could we test right now, and what would need expert help? With a few days training or coaching you could learn a lot, but for the purpose of this conference paper we will reduce our scope to OWASP web application vulnerability number two: *Broken Authentication and Session Management*. I've selected these not because they are easy to understand (the OWASP number one, *Injection* is pretty straightforward), but because they are actually rock-hard security testing areas that will clarify the divide between what any tester can do and what is best left to experts until your skills are sufficiently developed, if you choose to become an expert.

Let us start now by considering something that seems too trivial for thorough testing: The login screen. This is often the first area of functionality developed and simply tested by checking an access control matrix, i.e. a clerical role has restricted access to functionality and data, a manager has a larger subset, and an administrator has full access to everything.



Now let us look at this function from an attacker's point of view. If they can simply login to our system and start using it, they can start abusing it, especially if they can escalate their privileges to an administrator level. We need to test controls are effective in preventing attackers from logging in. We also need

to verify the associated functionality is as well protected as the login function. Let us do that by working through these tests:

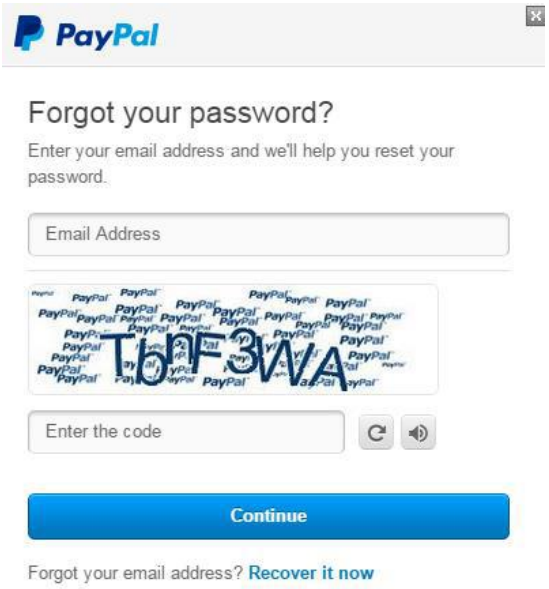
Bad passwords: If allowed passwords are very short, blank, match the username, use dictionary words or names, or the default password remains usable, the passwords will be discovered by attackers. Could you test the controls preventing those weaknesses are effective, and that the password complexity matches a password policy? Yes, you could. Just try entering each type of bad password.

Predictable usernames: Obtain several system-generated usernames in quick succession. Can you see any pattern in the names generated? If so, it will be possible for an attacker to guess existing and future usernames. That is bad. Raise a (security) defect!

Insecure Storage of Credentials: If usernames and passwords are stored in clear unencrypted text they will be found and used by attackers. Can you see them in the database? Get a DBA to help you check it out.

Non-unique usernames: If the system allows self-registration and specification of users own usernames, an attacker can use this functionality to register a username multiple times with different passwords and use the responses to determine if they have correctly guessed a valid existing username and password combination. Test to see if creating non-unique usernames is possible. If so, can users also choose the same password and access each other's accounts? If they cannot choose the same password, the response effectively reveals the password of the other user with the same name. It's all bad. Raise a defect!

Talkative Failure Messages: If you enter a valid username and invalid password, does the error message tell you the password was incorrect? Now enter an invalid username and valid password, does the error message tell you the username was incorrect? If so to either of these tests, then the guesswork of the attacker has been halved. It is a security fail and you can prove it.

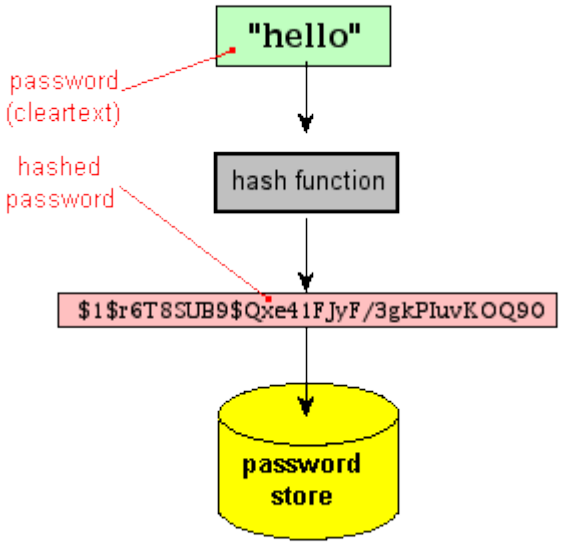


Forgotten Password: Is it possible to try unlimited uses of this function to discover valid usernames? If you must answer a secondary challenge instead of the main login, would it be easier to guess than a password? Can users set their own weak password recovery challenges such as 'do I own a house?', or obvious password hints such as the actual password?

Can the user specify the email address to send a unique recovery URL at the time of the challenge instead of initial registration? Does the application allow the user into an authenticated session straight after passing the challenge, without ever entering the password? Does the application reveal the password after passing the challenge, without requiring the password to change (allowing indefinite use without detection)? Does

the application allow users to reset their password without sending an email notification to the registered user? If the answer to any of these tests is yes, we have a security problem.

Password Change: Can users immediately change their passwords if they are compromised? If so, that is good, but it must be done securely. If attackers can access the password change function without going through the authentication process first, they may bypass stronger defences elsewhere. Repeat the test for talkative failure messages here, and for unlimited 'existing password' guesses. If you deliberately mismatch the 'new password' and 'confirm new password' fields, does the response indicate you guessed the existing password correctly? Ask a penetration tester to examine the HTTP response in detail, as an attacker would do, to look for that weakness. Also ask them to use an intercepting proxy to supply a different username to the current user in the username field to find out if it can be over-ridden. Alternatively install a free intercepting proxy such as Burp Suite or Zap and try it yourself.



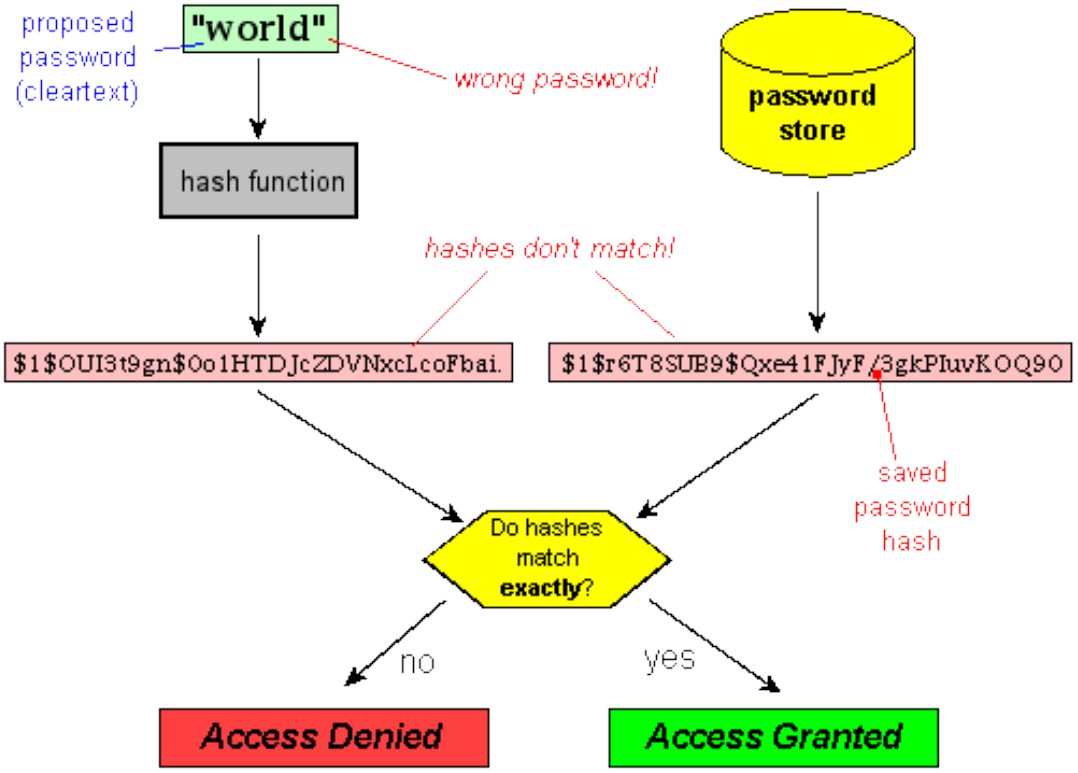
Incomplete validation of credentials: Even the title sounds difficult, but is it really? Consider a new user registering their password for the first time. To this explanation we will allow a non-complex all-lowercase text password such as '*hello*'. As explained earlier, user passwords should never be stored in the clear because attackers will find and use them. Instead of storing as plain text, the system will convert the password to a fixed length encrypted string (a.k.a. fingerprint) using a hashing algorithm. To ensure every hash is unique, each username has an associated 'salt' which is a unique key to the hashing algorithm used only by that username. Even if multiple users chose the same password, their hash values would not be the same.

Password hashing is one-way encryption, it is not possible to work backwards from a hash to the original password value. If a system can remind you of your original password it is not using hashing and is vulnerable. If any part of the input changes, the resulting hash is totally different as shown below:

hash("hello") = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824

hash("hello") = 58756879c05c68dfac9866712fad6a93f8146f337a69afe7dd238f3364946366

Every subsequent time the user logs into the system the password entered (e.g. *world*) is hashed using the same algorithm and salt as the first time, and the generated hash is compared with stored password hash. If they match, the user is allowed in, if they do not match the user is prevented from passing the login screen.



There are several ways hackers can crack the security provided by hashed passwords:

1) Dictionary attacks: Using common passwords, phrases, words and other strings such as 'leet speak' ("hello" becomes "h3110") to try guessing a password, then hashing every word in the list and comparing them to the hash values in the genuine password file when the security breach is underway.
2) Brute-Force attacks: Try every possible combination of characters up to a given length. These attacks are computationally expensive and inefficient but will always find the password eventually. The best defences make searching through all possible strings take too long to be worthwhile.
3) Lookup Tables: Pre-compute the hashes of passwords and compare them with hundreds of genuine password hashes per second. Try it against your own password by hashing it here: http://www.hashemall.com/ then using a free hash cracker https://crackstation.net/
4) Reverse Lookup Tables: Create a lookup table that maps each password hash from the compromised user account database to a list of users who had that hash, since many often have the same passwords. The attacker then hashes each password guess and uses the lookup table to get a list of users whose password was the attacker's guess.
5) Rainbow tables: Reduce the size of lookup tables and store more hashes in the same amount of space. Saves memory but reduces cracking speed.

You might think the odds of finding matches between guessed password hashes and a database of genuine password hashes would be very long but the 'birthday paradox' proves that wrong. How many people do you think must be in the same room as you for the chance to be greater than even that another person has the same birthday as you? Answer = 253. How many people must be in the same room for the chance to be greater than even that at least two people share the same birthday? Answer = 23.

In the first instance you are looking for someone with a specific birthday date that matches your own. In the second instance, you are looking for any two people who share the same birthday. There is a higher probability of finding two people who share a birthday than finding another person who shares your birthday. Hackers tend to look for any two matching password hashes (their guesses and any password database hash) rather than persistently trying to brute-force one hash value.

Now we need to test the developers are properly validating the passwords. Sometimes, too often in fact, developers reduce the length of the password used by the hashing algorithm, ignore upper- and lower-case differences, and omit special characters because they are unsure how to defend against code injection. All these mistakes make it harder for users to protect their accounts with long and strong passwords and make it easier for attackers to discover the passwords. You can test this by creating a maximum length password then trying to login without the last password character. If that works, keep omitting characters until you discover how many are really used by the hashing algorithm. Then test if upper- and lower-case passwords are validated by trying them without applying the case used when the password was created. Finally test if every type of unusual character (< >' " ; : & % etc.) is being validated by creating passwords that contain unusual characters but missing them out when you try logging in afterwards.

Now you tell penetration testers which authentication controls you have already tested, and direct them to testing the more technical vulnerabilities: *Predictable initial passwords, insecure distribution of credentials, fail-open login mechanisms, vulnerable credentials transmission, 'remember me' functionality, user impersonation functionality, multi-stage login defects, and brute-forcible login*.

If we can do the same thing for session management, we could apply cost effective security testing by leaving the tasks that would take a long time to learn in the hands of experts, while

taking care of the tasks that would be an inefficient use of expert time if they can be handled by system testers. Let us give it a try.

HTTP is stateless. Each request-response message pair is an independent transaction. Dynamic web-application functionality requires a SESSION to link user requests. Typically, this is implemented by issuing each user a unique session token which is resubmitted by the user to link sequences of requests. If an attacker can capture and use another users' session, they can completely by-pass the authentication mechanism. The longer a session token remains active, the greater the window of opportunity for an attacker to capture, guess, or misuse a valid token.

Disclosure of session tokens in the logs: Can you see session tokens in the logs or the URL query string (in which case the token will probably appear in users' browser logs, web server logs, corporate or ISP proxy server logs, reverse proxy logs, or Referrer logs of any application users visit by following off-site links)? If yes, it's a security fail.

With help from a penetration tester and an intercepting proxy tool, you could validate *vulnerable session termination, weak session token generation, weak session token handling, disclosure of tokens, encrypted token vulnerabilities, client exposure to token hijacking, liberal cookie scope, meaningful tokens and predictable session tokens*. You probably do not believe me so let's try those last two in the list.

Set-Cookie: ASP.NET_SessionId=75 73 65 72 3d 64 65 63 6c 61 6e 3b 61 70 70 3d 61 64 6d 69 6e 3b 64 61 74 65 3d 30 35 2f 30 37 2f 32 30 31 35

The session token above initially looks like a long string of meaningless characters. Let us look a little closer. There are many digits and some letters. None of the letters are higher in the alphabet than the letter 'f'. This suggests it could be a hexadecimal string. Copy the string into a hexadecimal to text converter and this is what we see:

user=declan;app=admin;date=05/07/2015

Now we know it would be easy to create a valid session token in this case, probably by adjusting the date and converting text to hexadecimal. Session tokens might be less obviously meaningful than the example above, but penetration testers have tools to assist the analysis.

Next, we have an implausibly difficult security test: Predictable session tokens. The weaknesses that attackers will be looking to exploit are - concealed sequences, weak random number generation, and time dependencies. Check out the numbers shown below which represent captured session tokens:

56543-1424798254115

56544-1424798303925

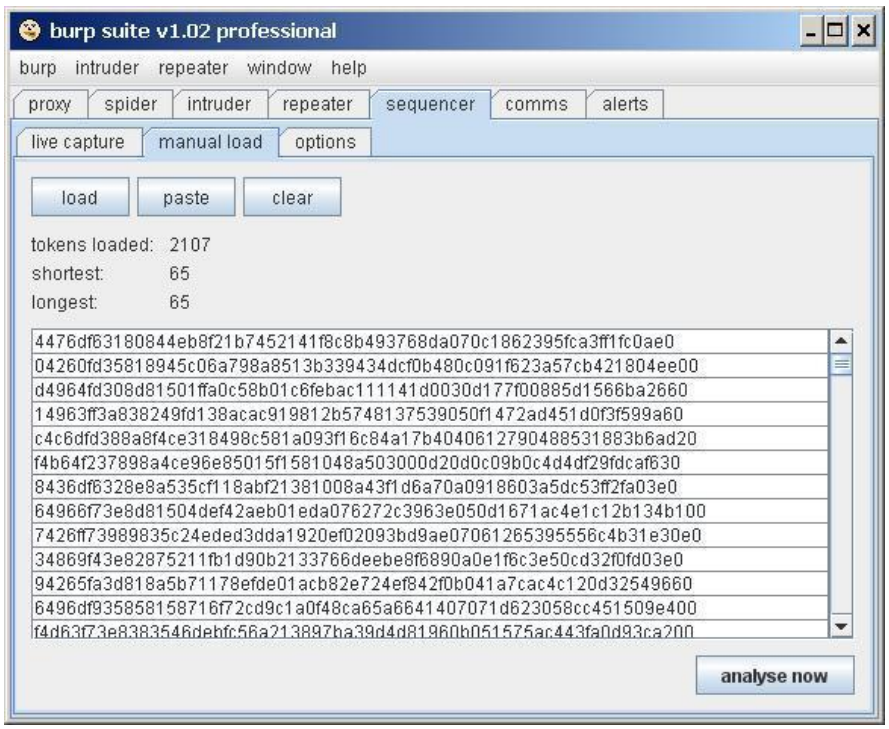
?

56546-1424798337916

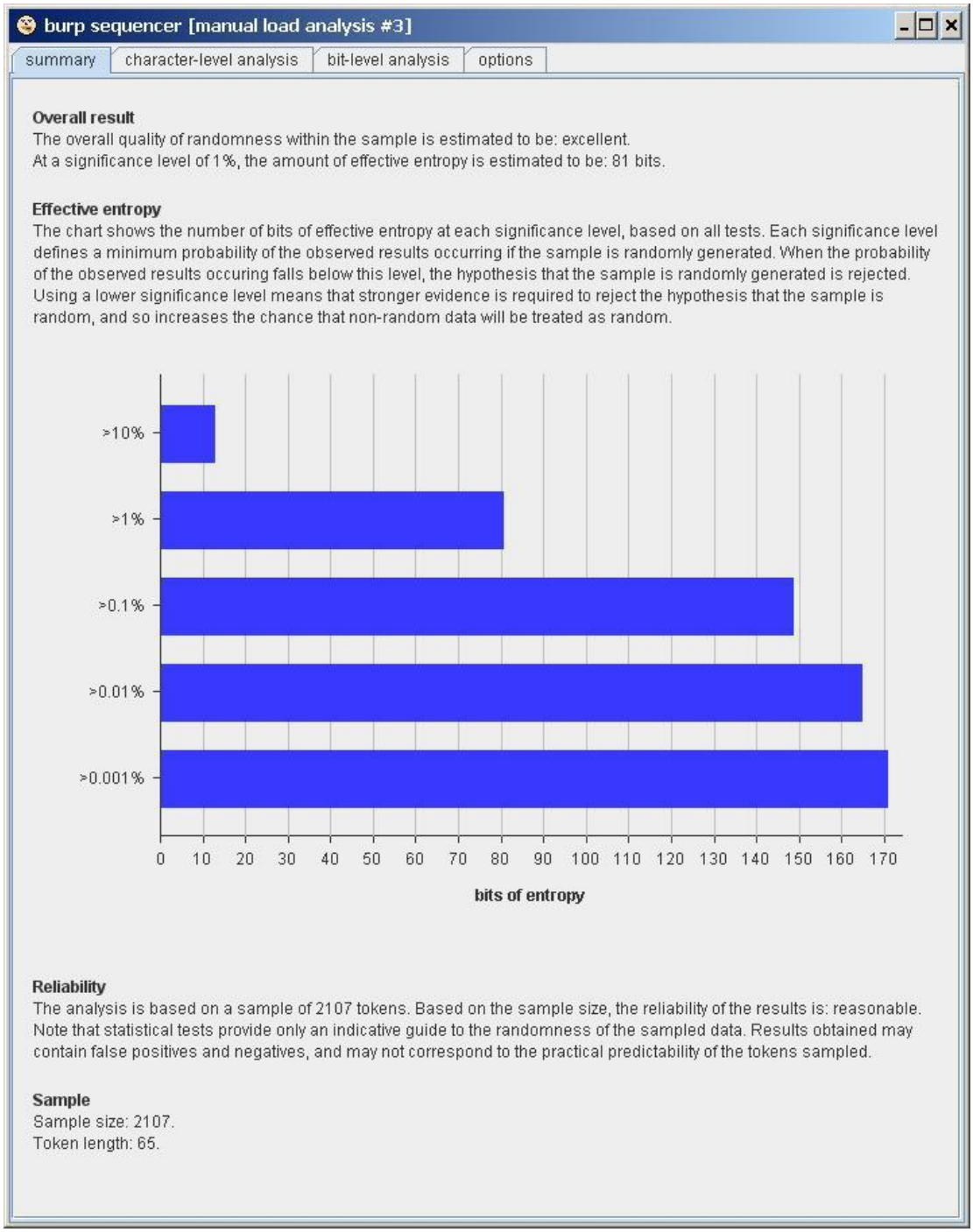
The first component is an incrementing sequence.

The second component is the time in milliseconds.

The missing value was issued to another user and can be predicted / brute forced within the range of possibilities. The first part will be 56545-14247983. The second part will be a time in milliseconds in the range of 03926 to 37915. Tell a penetration tester to determine how predictable session tokens are and they will use tools such as Burp Suite sequencer to perform the analysis.



The overall result of analysing a sample of tokens will be a randomness (effective entropy) score. The lower the probability the observed results are random, the easier it becomes for an attacker to predict a session token and by-pass authentication.



That is about as tough as security testing gets. You do not have to become an expert and undertake all the tests but understanding what the experts are doing puts you on much firmer ground for mitigating security risks and spending test resources efficiently. Here is a quick recap:

1. **Security testing skills are within the project team capability.**
2. **Recognizing which security tests, you can do now will save money for expert help.**
3. **Effectively manage the experts who are helping you to test the difficult tasks.**

Good luck!