Web Technology
Unit 5
Web Security and Authentication

By Dr Prasanna B T
SJCE,JSSSTU,MysuruUNIT 5

Security Principles

It is often the case that a developer will only consider security toward the end of a project, and by then it is much too late. Errors in the hosting configuration, code design, policies, and implementation can infiltrate through the application like holes in Swiss cheese. Filling these holes takes time, and the patched systems are often less elegant and manageable, if the holes get filled at all. The right way of addressing security is right from the beginning and all along the way so that you can plan for a secure system and hopefully have one in the end. Security theory and practice will guide you in that never-ending quest to proactively defend your data and systems, which you will see, touches all aspects of software development.

The principal challenge with security is that threats exist in so many different forms. Not only is a malicious hacker on a tropical island a threat but so too is a sloppy programmer, a disgruntled manager, or a naive secretary. Moreover, threats are ever changing, and with each new counter measure, new threats emerge to supplant the old ones. Since websites are an application of networks and computer systems, you must draw from those disciplines to learn many foundational security ideas. Later you will also see some practical ways to harden your system against malicious users and defend against programming errors.

Information Security

There are many different areas of study that relate to security in computer networks. Information security is the holistic practice of protecting information from unauthorized users. Computer/IT security is just one aspect of this holistic thinking, which addresses the role computers and networks play. The other is information assurance, which ensures that data is not lost when issues do arise.

The CIA Triad

At the core of information security is the CIA triad: *confidentiality*, *integrity*, and *availability*, often depicted with a triangle showing their equality as in Figure .

FIGURE The CIA triad: confidentiality, integrity, and availability

Confidentiality is the principle of maintaining privacy for the data you are storing, transmitting, etc. This is the concept most often thought of when security is brought up.

Integrity is the principle of ensuring that data is accurate and correct. This can include preventing unauthorized access and modification, but also extends to disaster preparedness and recovery.

Availability is the principle of making information available when needed to authorized people. It is essential to making the other two elements relevant, since without it, it's easy to have a confidential and integral system (a locked box). This can be extended to high-availability, where redundant systems must be in place to ensure high uptime.

Risk Assessment and Management

The ability to assess risk is crucial to the web development world. Risk is a measure of how likely an attack is, and how costly the impact of the attack would be if successful. In a public setting like the WWW, any connected computer can attempt to attack your site, meaning there are potentially several million threats. Knowing which ones to worry about allows you to identify the greatest risks and achieve the most impact for your effort by focusing on them.

Actors, Impact, Threats, and Vulnerabilities

Risk assessment uses the concepts of actors, impacts, threats, and vulnerabilities to determine where to invest in defensive countermeasures.

The term "actors" refers to the people who are attempting to access your system. They can be categorized as internal, external, and partners.

- Internal actors are the people who work for the organization. They can be anywhere in the organization from the cashier through the IT staff, all the way to the CEO. Although they account for a small percentage of the attacks, they are especially dangerous due to their internal knowledge of the systems.
- External actors are the people outside of the organization. They have a wide range of intent and skill, and they are the most common source of attacks. It turns out that more than three quarters of external actors are affiliated with organized crime or nation states.[1]
- Partner actors are affiliated with an organization that you partner or work with. If your partner is somehow compromised, there is a chance your data is at risk as well because quite often partners are granted some access to each other's systems (to place orders, for example).

The impact of an attack depends on what systems were infiltrated and what data was stolen or lost. The impact relates back to the CIA triad since impact could be the loss of availability, confidentiality, and/or integrity.

- A *loss of availability* prevents users from accessing some or all of the systems. This might manifest as a denial of service attack, or a SQL injection attack (described later), where the payload removes the entire user database, preventing logins from registered users.
- A *loss of confidentiality* includes the disclosure of confidential information to a (often malicious) third party. It can impact the human beings behind the usernames in a very real way, depending on what was stolen. This could manifest as a cross-site script attack where data is stolen right off your screen or a full-fledged database theft where credit cards and passwords are taken.
- A *loss of integrity* changes your data or prevents you from having correct data. This might manifest as an attacker hijacking a user session, perhaps placing fake orders or changing a user's home address.

A threat refers to a particular path that a hacker could use to exploit a vulnerability and gain unauthorized access to your system. Sometimes called attack vectors, threats need not be malicious. A flood destroying your data center is a threat just as much as malicious SQL injections, buffer overflows, denial of service, and cross-site scripting attacks.

Broadly, threats can be categorized using the STRIDE mnemonic, developed by Microsoft, which describes six areas of threat:[2]

- Spoofing – The attacker uses someone else's information to access the system.
- Tampering – The attacker modifies some data in nonauthorized ways.
- Repudiation – The attacker removes all trace of their attack, so that they cannot be held accountable for other damages done.
- Information disclosure – The attacker accesses data they should not be able to.
- Denial of service – The attacker prevents real users from accessing the systems.
- Elevation of privilege – The attacker increases their privileges on the system thereby getting access to things they are not authorized to do.

Vulnerabilities are the security holes in your system. This could be an unsanitized user input or a bug in your Apache software, for example. Once vulnerabilities are identified, they can be assessed for risk. Some vulnerabilities are not fixed because they are unlikely to be exploited, while others are low risk because the consequences of an exploit are not critical. The top five classes of vulnerability from the Open Web Application Security Project[3] are:

1. Injection
2. Broken authentication and session management
3. Cross-site scripting
4. Insecure direct object references
5. Security misconfiguration

Security Policy

One often underestimated technique to deal with security is to clearly articulate policies to users of the system to ensure they understand their rights and obligations. These policies typically fall into three categories:

- Usage policy defines what systems users are permitted to use, and under what situations. A company may, for example, prohibit social networking while at work, even though the IT policies may allow that traffic in. Usage policies are often designed to reduce risk by removing some attack vector from a particular class of system.
- Authentication policy controls how users are granted access to the systems. These policies may specify where an access badge is needed, a biometric ID, or when a password will suffice. Often hated by users, these policies most often manifest as simple password policies, which can enforce length restrictions and alphabet rules as well as expiration of passwords after a set period of time.
- Legal policies define a wide range of things including data retention and backup policies as well as accessibility requirements (like having all public communication well organized for the blind). These policies must be adhered to in order to keep the organization in compliance.

Business Continuity

The unforeseen happens. Whether it's the death of a high-level executive, or the failure of a hard drive, business must continue to operate in the face of challenges. The best way to be prepared for the unexpected is to plan while times are good and thinking is clear in the form of a business continuity plan/disaster recovery plan. These plans are normally very comprehensive and include matters far beyond IT. Some considerations that relate to IT security are as follows.

Admin Password Management

If a bus suddenly killed the only person who has the password to the database server, how would you get access? This type of question may seem morbid, but it is essential to have an answer to it. The solution to this question is not an easy one since you must balance having the passwords available if needed and having the passwords secret so as not to create vulnerability.

There must also be a high level of trust in the system administrator since they can easily change passwords without notifying anyone, and it may take a long time until someone notices. Administrators should not be the only ones with keys, as was the case in 2008 when City of San Francisco system administrator, Terry Childs, locked out his own employer from all the systems, preventing access to anyone but himself.[5]

Some companies include administrator passwords in their disaster recovery plans. Unfortunately, those plans are often circulated widely within an organization, and divulging the root passwords widely is a terrible practice.

A common plan is a locked envelope or safe that uses the analogy of a fire alarm—break the seal to get the passwords in an emergency. Unfortunately, a sealed envelope is easily opened and a locked safe can be opened by anyone with a key (single-factor authentication). To ensure secrecy, you should require two people to simultaneously request access to prevent one person alone from secretly getting the passwords in the box, although all of this depends on the size of the organization and the type of information being secured.

Backups and Redundancy

Backups are an essential element of business continuity and are easy to do for web applications so long as you are prepared to do them. What do you typically need to back up? The answer to this question can be determined by first deciding what is required to get a site up and running:

- A server configured with Apache to run our PHP code with a database server installed on the same or another machine.
- The PHP code for the domain.
- The database dump with all tables and data.

The speed with which you want to recover from a web breach determines which of the above you should have on hand. For the fastest response, a live backup server with everything already mirrored is the best approach, but this can be a costly solution.

In less critical situations, simply having the database and code somewhere that is accessible remotely might suffice. Any downtime that occurs while the server is reconfigured may be acceptable, especially if no data is lost in the process.

No matter the speed you wish to recover, backups can be configured to happen as often as needed, with a wide range of options (full vs. differential). You must balance backup frequency against the value of information that would be lost, so that critical information is backed up more frequently than less critical data.

Geographic Redundancy

The principle of a geographically distinct backup is to have backups in a different place than the primary systems in case of a disaster. Storing CD backups on top of a server does you no good if the server catches fire (and the CDs with it). Similarly, having a backup server in the same server rack as the primary system makes them prone to the same outages. When this idea is taken to a logical extreme, even a data center in the same city could be considered nonsecure, since a natural disaster or act of war could impact them both.

Thankfully, purchasing geographically remote server and storage space can be done relatively cheaply using a shared hosting environment. Look for hosts that tell you the geographic locations of their servers so that you can choose one that is geographically distinct from your primary systems.

Stage Mock Events

All the planning in the world will go to waste if no one knows the plan, or the plan has some fatal flaws. It's essential to actually execute mock events to test out disaster recovery plans. When planning for a mock disaster scenario, it's a perfect time to "kill" some key staff by sending them on vacation allowing new staff to get up to speed during the pressure of a mock disaster. In addition to removing staff, consider removing key pieces of technology to simulate outages (take away phones, filter out Google, take away a hard drive). Problems that arise in the recovery of systems during a mock exercise provide insight into how to improve your planning for the next scenario, real or mock. It can also be a great way to cross-train staff and build camaraderie in your teams.

Auditing

Auditing is the process by which a third party is invited (or required) to check over your systems to see if you are complying with regulations and your claims. Auditing happens in the financial sector regularly, with a third-party auditor checking a company's financial records to ensure everything is as it should be. Oftentimes, simply knowing an audit will be done provides incentive to implement proper practices.

The practice of logging, where each request for resources is stored in a secure log, provides auditors with a wealth of data to investigate. Linux, by default, stores logs related to `ssh` and other network access. You can exert control over these and the logging of your Apache server.

Another common practice is to use databases to track when records are edited or deleted by storing the timestamp, the record, the change, and the user who was logged in. These logs are often stored in separate, audit tables.

Secure by Design

Secure by design is a software engineering principle that tries to make software better by acknowledging and addressing that there are malicious users out there. By continually distrusting user input (and even internal values) throughout the design and implementation phases, you will produce more secure software than if you didn't consider security at every stage. Some techniques that have developed to help keep your software secure include code reviews, pair programming, security testing, and security by default.

Figure illustrates how security can be applied at every stage of the classic waterfall software development life cycle. While not all of the illustrated inputs are covered in this textbook, it does cover many of the most impactful strategies for web development.

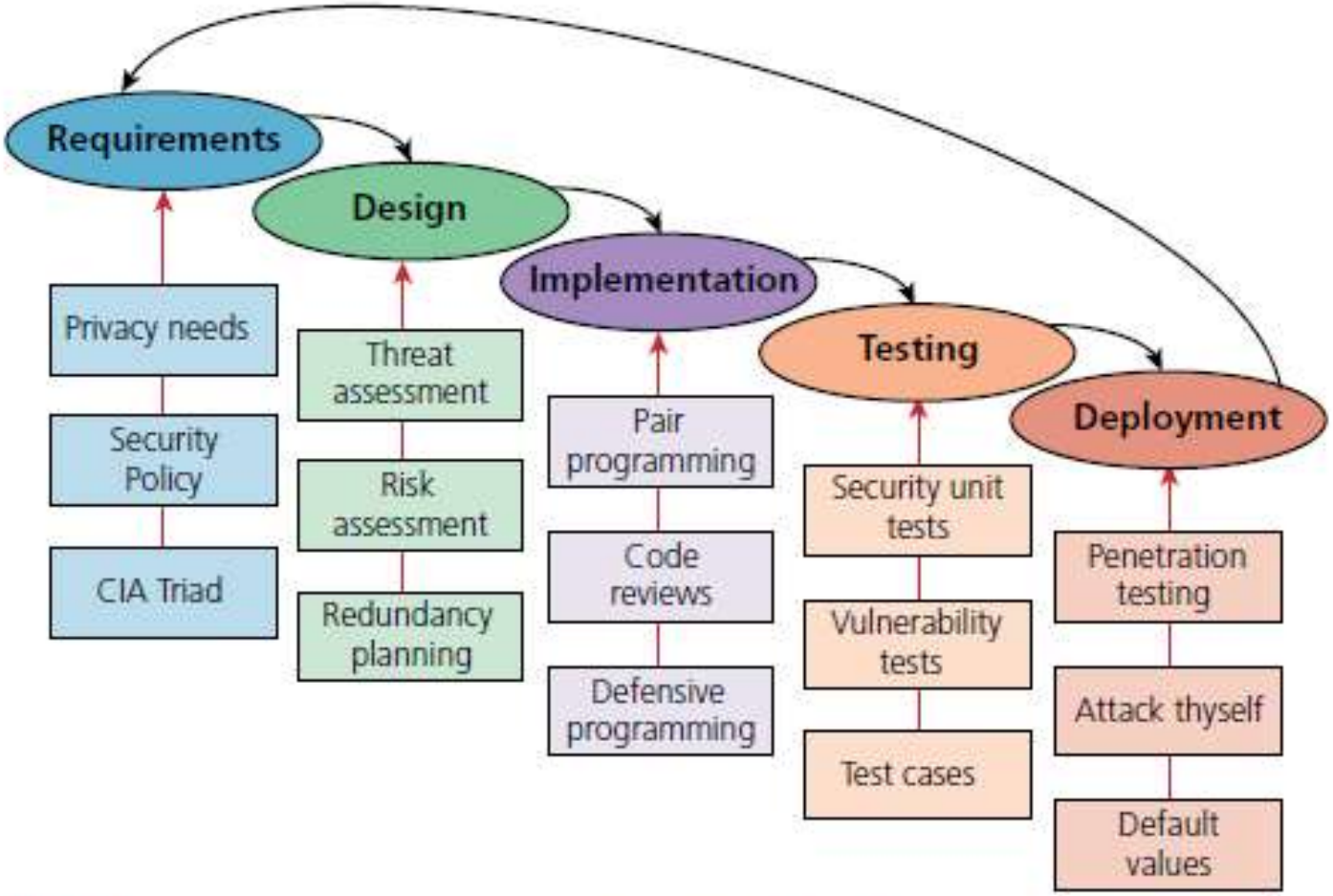


FIGURE Some examples of security input into the SDLC

Code Reviews

In a code review system, programmers must have their code peer-reviewed before committing it to the repository. In addition to peer-review, new employees are often assigned a more senior programmer who uses the code review opportunities to point out inconsistencies with company style and practice.

Code reviews can be both formal and informal. The formal reviews are usually tied to a particular milestone or deadline whereas informal reviews are done on an ongoing basis, but with less rigor. In more robust code reviews algorithms can be traced or tested to ensure correctness.

Unit Testing

Unit testing is the principle of writing small programs to test your software as you develop it. Usually the *units* in a unit test are a module or class, and the test can compare the expected behavior of the class against the actual output. If you break any existing functionality, a unit test will discover it right away, saving you future headache and bugs. Unit tests should be developed alongside the main web application and be run with code reviews or on a periodic basis. When done properly, they test for boundary conditions and situations that can hide bugs, which could be a security hole.

Pair Programming

Pair programming is the technique where two programmers work together at the same time on one computer. One programmer *drives* the work and manipulates the mouse and keyboard while the other programmer can focus on catching mistakes and high-level *thinking*. After a set time interval the roles are switched and work continues. In addition to having two minds to catch syntax errors and the like, the team must also agree on any implementation details, effectively turning the process into a continuous code review.

Security Testing

Security testing is the process of testing the system against scenarios that attempt to break the final system. It can also include penetration testing where the company attempts to break into their own systems to find vulnerabilities as if they were hackers. Whereas normal testing focuses on passing user requirements, security testing focuses on surviving one or more attacks that simulate what could be out in the wild.

Secure by Default

Systems are often created with default values that create security risks (like a blank password). Although users are encouraged somewhere in the user manual to change those settings, they are often ignored, as exemplified by the tales of ATM cash machines that were easily reprogrammed by using the default password.[6] Secure by default aims to make the default settings of a software system secure, so that those type of breaches are less likely even if the end users are not very knowledgeable about security.

Social Engineering

Social engineering is the broad term given to describe the manipulation of attitudes and behaviors of a populace, often through government or industrial propaganda and/or coercion. In security circles software engineering takes on the narrower meaning referring to the techniques used to manipulate people into doing something, normally by appealing to their baser instincts.

Social engineering is the human part of information security that increases the effectiveness of an attack. No one would click a link in an email that said *click here to get a virus*, but they might click a link to *get your free vacation*. A few popular techniques that apply social engineering are phishing scams and security theater.

Phishing scams, almost certainly not new to you, manifest famously as the Spanish Prisoner or Nigerian Prince Scams.[7] In these techniques a malicious user sends an email to everyone in an organization about how their password has expired, or their quota is full, or some other ruse to make them feel anxious that they must act by clicking the link and providing their login information. Of course the link directs them to a fake site that looks like the authentic site, except for the bogus URL, which only some people will recognize.

While good defenses, in the form of spam filters, will prevent many of these attacks, good **policies** will help too, with users trained not to click links in emails, preferring instead to always type the URL to log in. Some organizations go so far as to set up false phishing scams that target their own employees to see which ones will divulge information to such scams. Those employees are then retrained or terminated.

Security theater is when visible security measures are put in place without too much concern as to how effective they are at improving actual security. The visual nature of these theatrics is thought to dissuade potential attackers. This is often done in 404 pages where a stern warning might read:

> *Your IP address is XX.XX.XX.XX. This unauthorized access attempt has been logged. Any illegal activity will be reported to the authorities.*

This message would be an example of security theater if this stern statement is a site's only defense. When used alone, security theater is often ridiculed as a serious technique, but as part of a more complete defense it can contribute a deterrent effect.

Authentication

To achieve both *confidentiality* and *integrity*, the user accessing the system must be who they purport to be. Authentication is the process by which you decide that someone is who they say they are and therefore permitted to access the requested resources. Whether getting entrance to an airport, getting past the bouncer at the bar, or logging into your web application, you have already seen authentication in action.

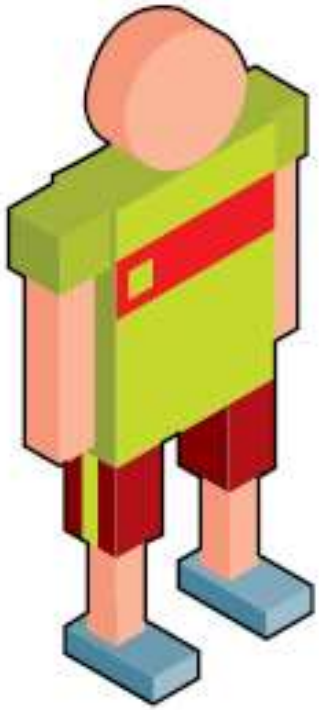
Authentication Factors

Authentication factors are the things you can ask someone for in an effort to validate that they are who they claim to be. As illustrated in Figure the three categories of authentication factor, knowledge, ownership, and inherence, are commonly thought of as *the things you know*, *the things you have*, and *the things you are*.

Knowledge

Knowledge factors are the things you know. They are the small pieces of knowledge that supposedly only belong to a single person such as a password, PIN, challenge question (what was your first dog's name), or pattern (like on some mobile phones).

These factors are vulnerable to someone finding out the information. They can also be easily shared.



What you know (Knowledge)

Passwords, PIN, security questions, ...

What you have (Ownership)

Access card, cell phone, cryptographic FOB, ...

What you are (Inherence)

Retinas, fingerprints, DNA, walking gait, ...

FIGURE Authentication factors

Knowledge

Knowledge factors are the things you know. They are the small pieces of knowledge that supposedly only belong to a single person such as a password, PIN, challenge question (what was your first dog's name), or pattern (like on some mobile phones).

These factors are vulnerable to someone finding out the information. They can also be easily shared.

Ownership

Ownership factors are the things that you possess. A driving license, passport, cell phone, or key to a lock are all possessions that could be used to verify you are who you claim to be.

Ownership factors are vulnerable to theft just like any other possession. Some ownership factors can be duplicated like a key, license, or passport while others are much harder to duplicate such as a cell phone or dedicated authentication token.

Inherence Factors

Inherence factors are the things you are. This includes biometric data like your fingerprints, retinal pattern, and DNA sequence but sometimes it includes things that are unique to you like a signature, vocal pattern, or walking gait.

These factors are much more difficult to forge, especially when they are combined into a holistic biometric scan.

Single-Factor Authentication

Single-factor authentication is the weakest and most common category of authentication system where you ask for only one of the three factors. An implementation is as simple as knowing a password or possessing a magnetized key badge to gain access.

Single-factor authorization relies on the strength of passwords and on the users being responsive to threats such as people looking over their shoulder during password entry as well as phishing scams and other attacks. This is why banks do not allow you to use your birthday as your PIN and websites require passwords with special characters and numbers. When better authentication confidence is required, more than one authentication factor should be considered.

Multifactor Authentication

Multifactor authentication is where two distinct factors of authentication must pass before you are granted access. This dramatically improves security, with any attack now having to address two authentication factors, which will require at least two different attack vectors. Typically one of the two factors is a knowledge factor supplemented by an ownership factor like a card or pass. The inherent factors are still very costly to implement although they can provide better validation.

The way we all access an ATM machine is an example of two-factor authentication: you must have both the knowledge factor (PIN) and the ownership factor (card) to get access to your account.

So well accepted are the concepts of multifactor authentication that they are referenced by the US Department of Homeland Security as well as the credit card industry, which publishes standards that require two-factor authentication to gain access to networks where card holder information is stored.[8]

Multifactor authentication is becoming prevalent in consumer products as well, where your cell phone is used as the ownership factor alongside your password as a knowledge factor.

Third-Party Authentication

Some of you may be reading this and thinking, *this is hard*. Authentication is easy when it's a username and password, but not so when you really consider it in depth (and just wait until you see how to store the credentials).

Fortunately, many popular services allow you to use their system to authenticate the user and provide you with enough data to manage your application. This means you can leverage users' existing relationships with larger services to benefit from *their* investment in authentication while simultaneously tapping into the additional services *they* support.

Third-party authentication schemes like OpenID and oAuth are popular with developers and are used under the hood by many major websites including Amazon, Facebook, Microsoft, and Twitter, to name but a few. This means that you can present your users with an option to either log in using your system, or use another provider.

OAuth

Open authorization (OAuth) is a popular authorization framework that allows users to use credentials from one site to authenticate at another site. It has matured from version 1.0 in 2007 to the newest specification (2.0) in 2012. A constant work in progress, the writers acknowledge that many "noninteroperable implementations" are likely.[9]

Therefore, we will cover the broad strokes of OAuth, leaving out the implementation details that would differ from provider to provider.

OAuth uses four user roles in its framework.

- The resource owner is normally the end user who can gain access to the resource (though it can be a computer as well).
- The resource server hosts the resources and can process requests using access tokens.
- The client is the application making requests on behalf of the resource owner.
- The authorization server issues tokens to the client upon successful authentication of the resource owner. Often this is the same as the resource server.

Before you begin to work with an OAuth provider, you typically register with their authorization servers to obtain cryptographically secure codes you will use so the authentication server can validate that you are who you claim to be when requesting authorization on behalf of users.

As shown in Figure . , websites that implement OAuth (clients) direct resource owners (users) to log in at the authorization server. After a successful login, the authorization server transmits one-time tokens to the user in the form of a redirect to the client, which ensures the authentication token gets to the client. The client, armed with this authentication code, can combine it with the secret obtained originally to authenticate and request an access token, which can then be used to access protected resources.

These tokens are not passwords, but rather strings that may contain user info, expiration date, and even cryptographic information. The details of the tokens are left up to the implementation, but generally relate to the assets and data of that user. Granular authorization options are often maintained by the resource server (you can read but not post, for example), but this is up to the implementation. This means that to actually build a functioning system, you will have to learn about several implementations and manage each one a little bit differently. That in-depth exercise is left to the reader.

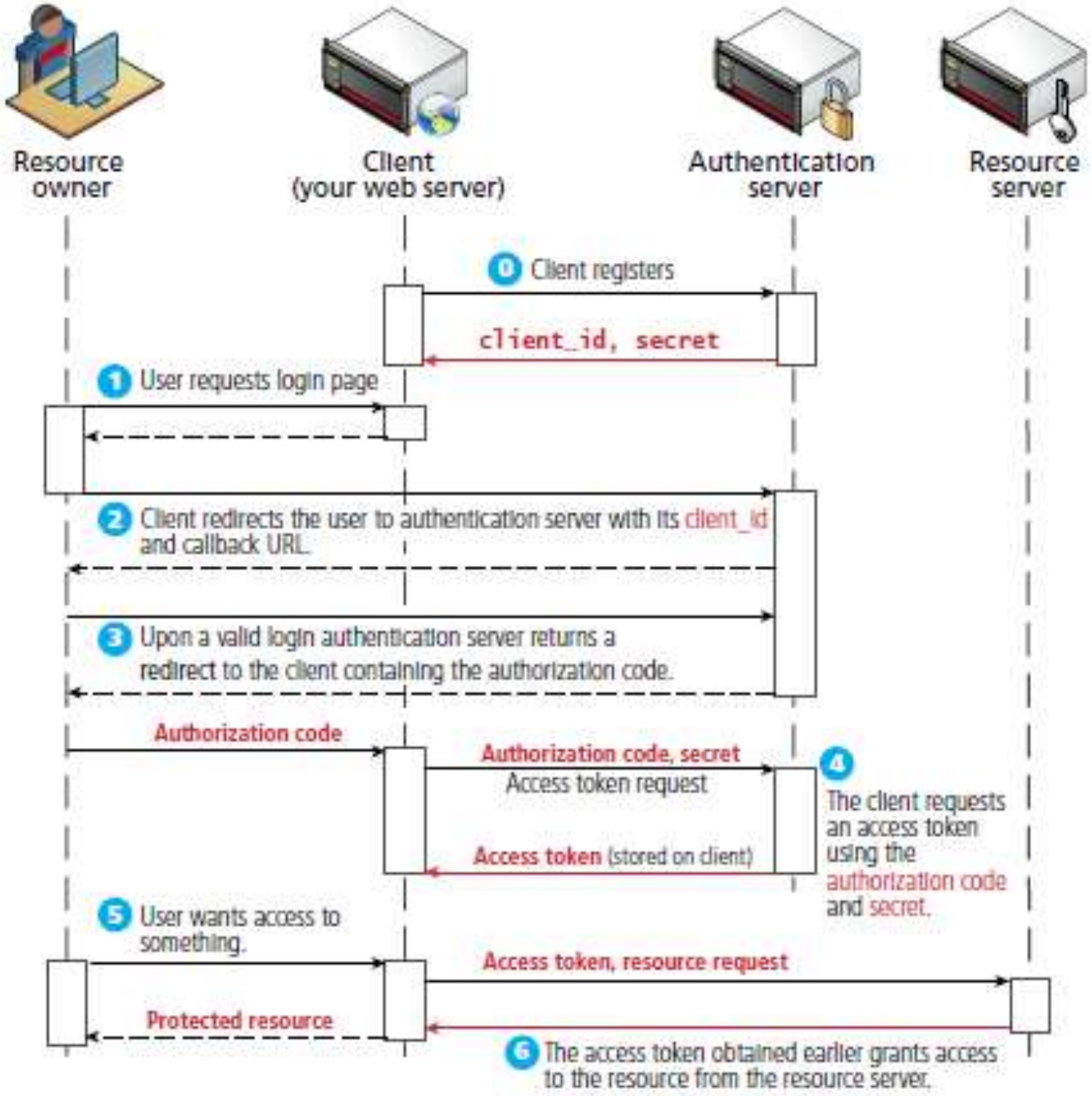


FIGURE The steps required to register and authenticate a user using OAuth

Authorization

Authorization defines what rights and privileges a user has once they are authenticated. It can also be extended to the privileges of a particular piece of software (such as Apache). Authentication and authorization are sometimes confused with one another, but are two parts of a whole. Authentication *grants* access, and authorization *defines* what the user with access can (and cannot) do.

The principle of least privilege is a helpful rule of thumb that tells you to give users and software only the privileges required to accomplish their work. It can be seen in systems such as Unix and Windows, with different privilege levels and inside of content management systems with complex user roles.

Starting out a new user with the least privileged account and adding permission as needed not only provides security but allows you to track who has access to what systems. Even system administrators should not use the root account for their day-to-day tasks, but rather escalate their privileges when needed.

Some examples in web development where proper authorization increases security include:

- Using a separate database user for read and write privileges on a database
- Providing each user an account where they can access their own files securely
- Setting permissions correctly so as to not expose files to unauthorized users
- Using Unix groups to grant users permission to access certain functionality rather than grant users admin access
- Ensuring Apache is not running as the root account (i.e., the account that can access everything)

Authorization also applies to roles within content management systems so that an editor and writer can be given authorization to do different tasks.

Hypertext Transfer Protocol Secure (HTTPS)

Now that you have a bit of understanding of the cryptography involved, the practical application of that knowledge is to apply encryption to your websites using the Hypertext Transfer Protocol Secure (HTTPS) protocol instead of the regular HTTP.

HTTPS is the HTTP protocol running on top of the Transport Layer Security (TLS). Because TLS version 1.0 is actually an improvement on Secure Sockets Layer 3.0 (SSL), we often refer to HTTP as running on TLS/SSL for compatibility reasons. Both TLS and SSL run on a lower layer than the application layer (back in Chapter 1 we discussed Internet Protocol and layers), and thus their implementation is more related to networking than web development. It's easy to see from a client's perspective that a site is secured by the little padlock icons in the URL bar used by most modern browsers (as shown in Figure).

An overview of their implementation provides the background needed to understand and apply secure encryption more thoughtfully. Once you see how the encryption works in the lower layers, everything else is just HTTP on top of that secure communication channel, meaning anything you have done with HTTP you can do with HTTPS.

FIGURE Screenshot from Google's Gmail service, using HTTPS

Secure Handshakes

The foundation for establishing a secure link happens during the initial handshake. This handshake must occur on an IP address level, so while you can host multiple secure sites on the same server, each domain must have its own IP address in order to perform the low-level handshaking as illustrated in Figure .

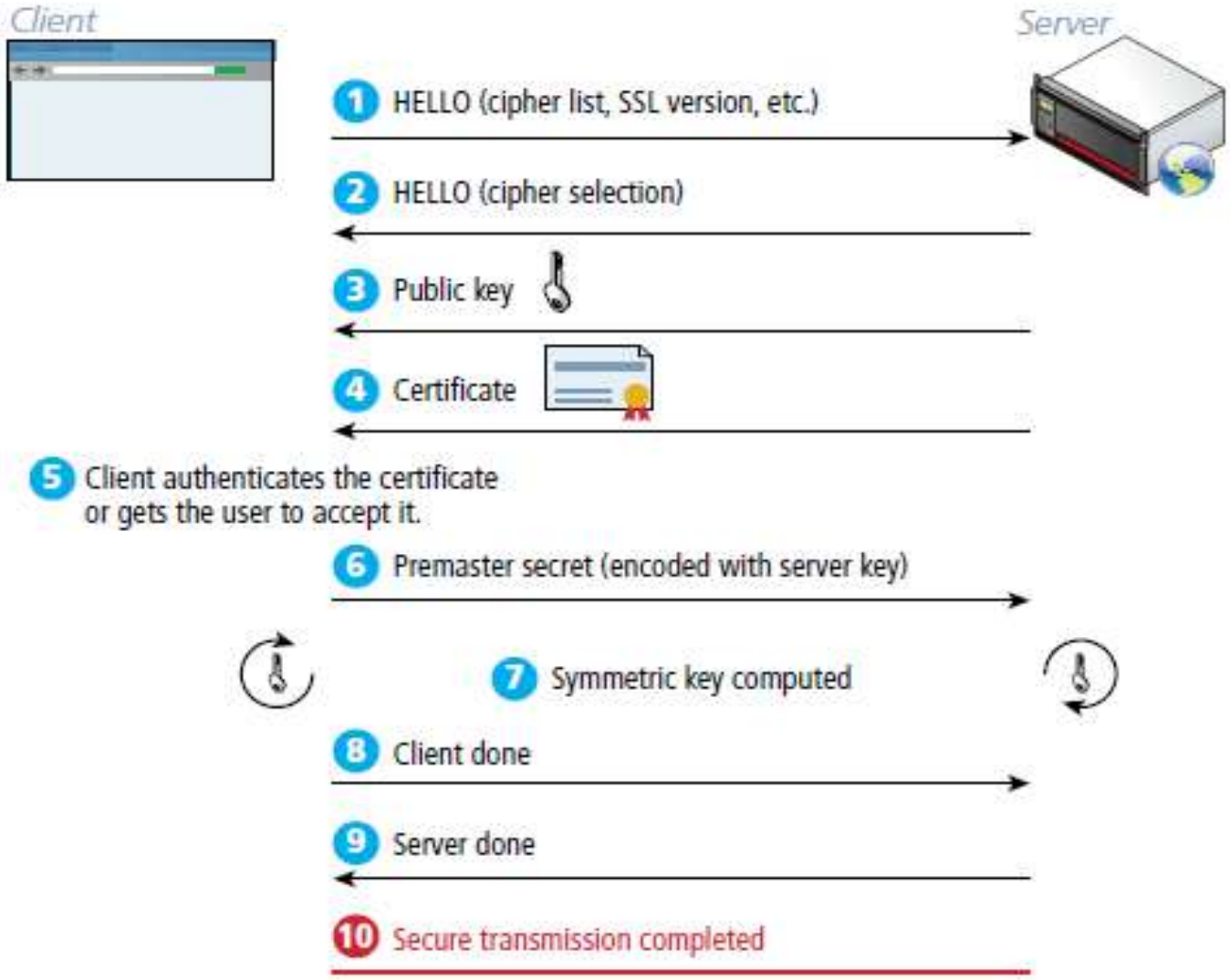


FIGURE SSL handshake

The client initiates the handshake by sending the time, and a list of cipher suites its browser supports to the server. The server, in response, sends back which of the client's ciphers it wants to use as well as a certificate, which contains information including a public key. The client can then verify if the certificate is valid. For self-signed certificates, the browser may prompt the user to allow an exception.

The client can then send a premaster secret (encrypted with the public key received from the server) back to the server. Using the random premaster secret both the client and server can compute a symmetric key. After a brief client message and server message declaring their readiness, all transmission can begin to be encrypted from here on out using the agreed-upon symmetric key.

Certificates and Authorities

The certificate that is transmitted during the handshake is actually an X.509 certificate, which contains many details including the algorithms used, the domain it was issued for, and some public key information. The complete X.509 specification can be found in the International Telecommunication Union's directory of public key frameworks.[14] A sample of what's actually transmitted is shown in Figure .

The certificate contains a signature mechanism, which can be used to validate that the domain is really who they claim to be. This signature relies on a third party to sign the certificate on behalf of the website so that if we trust the signing party, we can assume to trust the website.

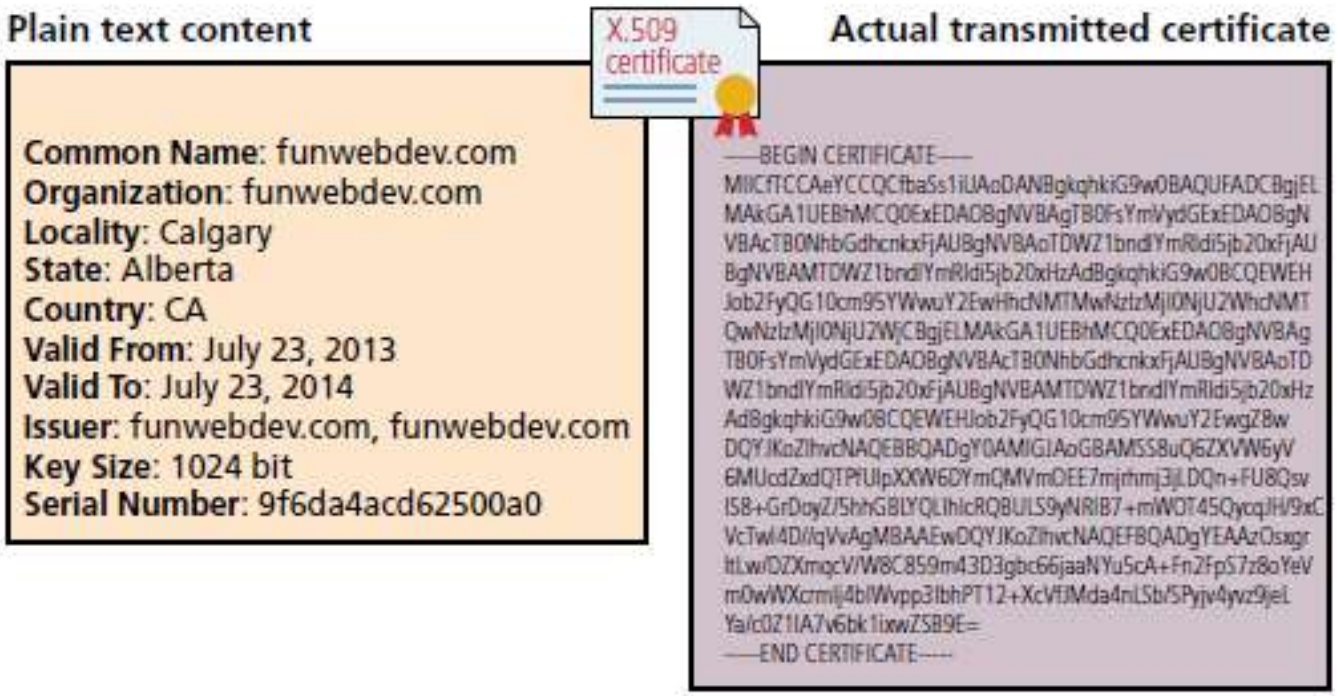


FIGURE The contents of a self-signed certificate for funwebdev.com

Certificate Authority

A Certificate Authority (CA) allows users to place their trust in the certificate since a trusted, independent third party signs it. The CA's primary role is to validate that the requestor of the certificate is who they claim to be, and issue and sign the certificate containing the public keys so that anyone seeing them can trust they are genuine.

In browsers, there are many dozens of CAs trusted by default as illustrated in Figure . A certificate signed by any of them will prevent the warnings that appear for self-signed certificates and in fact increase the confidence that the server is who they claim to be.

A signed certificate is essential for any website that processes payment, takes a booking, or otherwise expects the user to trust that the site is genuine.

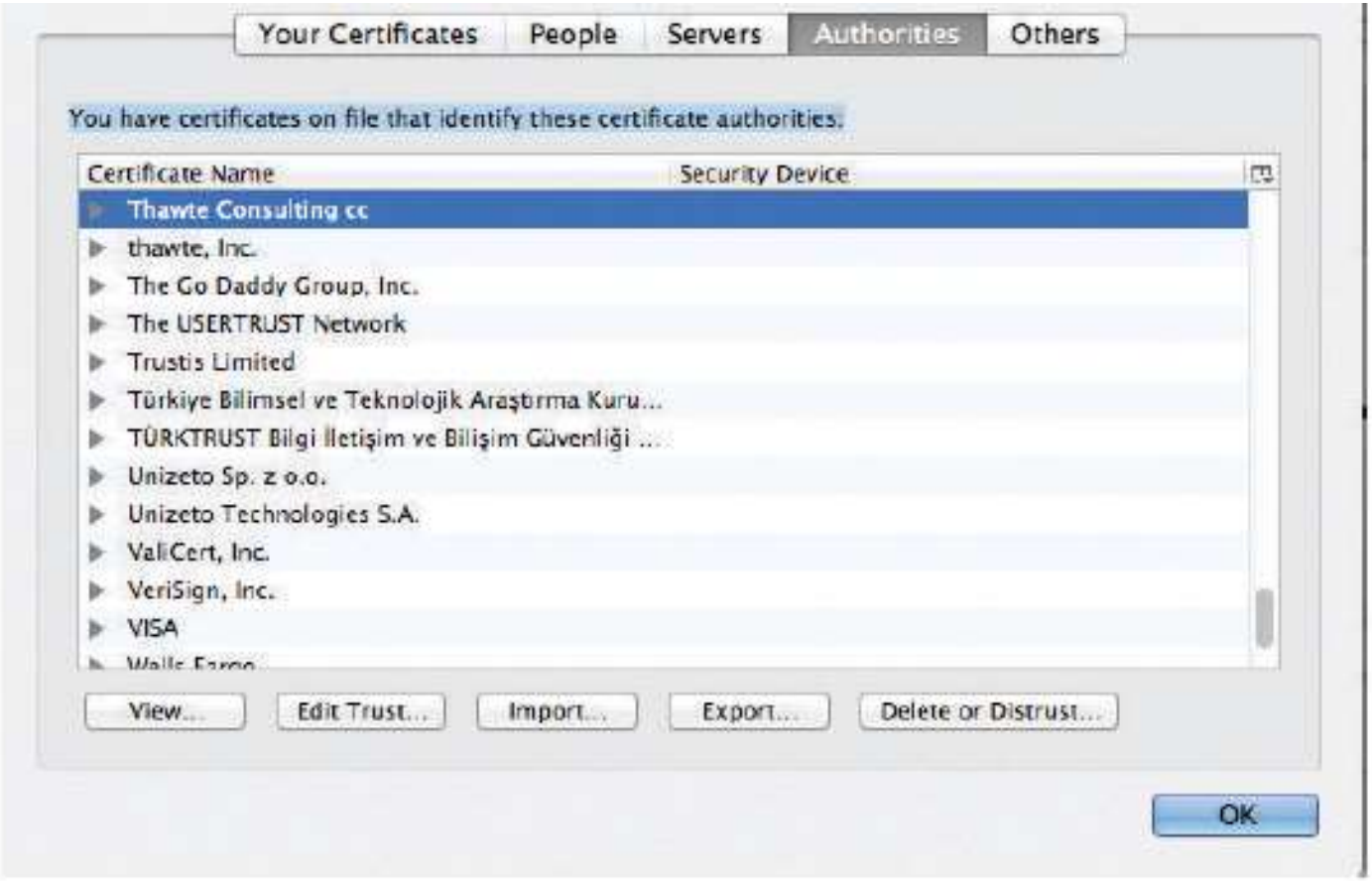


FIGURE The Firefox Certificate Authority Management interface

Self-Signed Certificates

An alternative to paying a Certificate Authority is to sign the certificates yourself. Self-signed certificates provide the same level of encryption, but the validity of the server is not confirmed. These are useful for development and testing environments, but not normally in production.

The downside of a self-signed certificate is that we are not leveraging the trust of the user (or browser) in known certificate authorities. Most browsers will warn users that your site is not completely secure as illustrated in the screen grab for funwebdev.com in Figure . Since users are not certain exactly what they are being told, they may lose faith that your site is secure and leave, making a signed certificate essential for any serious business.

FIGURE Firefox warning that arises from a self-signed certificate

Security Best Practices

With all our previous discussion of security thinking, cryptographic principles, and authentication in mind, it's now time to discuss some practical things you can do to harden your system against attacks.

A system will be targeted either purposefully or by chance. The majority of attacks are opportunistic attacks where a scan of many systems identifies yours for vulnerabilities. Targeted attacks occur less often, but are by their nature more difficult to block. Either way there are some great techniques to make your system less of a target.

Data Storage

With a good grasp of the authentication schemes and factors available to you, there is still the matter of what you should be storing in your database and server. It turns

UserID (int)	Username (varchar)	Password (varchar)
1	ricardo	password
2	randy	password

TABLE Plain Text Password Storage

out even household names like Sony,[15] Citigroup,[16] and GE Money[17] have had their systems breached and data stolen. If even globally active companies can be impacted, you must ask yourself: when (not if) you are breached, what data will the attacker have access to?

A developer who builds their own password authentication scheme may be blissfully unaware how this custom scheme could be compromised. The authors have seen students very often create SQL table structures similar to that in Table and code like that in Listing , where the username and password are both stored in the table. Anyone who can see the database can see all the passwords (in this case users `ricardo` and `randy` have both chosen the terrible password `password`).

This is dangerous for two reasons. Firstly, there is the *confidentiality* of the data. Having passwords in plain text means they are subject to disclosure. Secondly,

there is the issue of internal tampering. Anyone inside the organization with access to the database can steal credentials and then authenticate as another user, thereby compromising the *integrity* of the system and the data.

```php
//Insert the user with the password
function insertUser($username,$password){
   $link = mysqli_connect("localhost", "my_user", "my_password",
                          "Login");
   $sql = "INSERT INTO Users(Username,Password)
           VALUES('$username','$password')";
   mysqli_query($link, $sql);            //execute the query
}

//Check if the credentials match a user in the system
function validateUser($username,$password){
   $link = mysqli_connect("localhost", "my_user", "my_password",
                          "Login");
   $sql = "SELECT UserID FROM Users WHERE Username='$username' AND
           Password='$password'";
   $result = mysqli_query($link, $sql);   //execute the query
   if($row = mysqli_fetch_assoc($result)){
      return true;   //record found, return true.
   }

   return false; //record not found matching credentials, return false
}
```

LISTING PHP functions to insert and select a record with plaintext storage

Secure Hash

Instead of storing the password in plain text, a better approach is to store a hash of the data, so that the password is not discernable. One-way hash functions are algorithms that translate any piece of data into a string called the digest. You may have used hash functions before in the context of hash tables. Their one-way nature means that although we can get the digest from the data, there is no reverse function to get the data back. In addition to thwarting hackers, it also prevents malicious users from casually browsing user credentials in the database.

Cryptographic hash functions are one-way hashes that are cryptographically secure, in that it is virtually impossible to determine the data given the digest. Commonly used ones include the Secure Hash Algorithms (SHA)[18] created by the US National Security Agency and MD5 developed by Ronald Rivest, a cryptographer from MIT.[19] In our PHP code we can access implementations of MD5 and SHA through the `md5()` or `sha1()` functions. MySQL also includes implementations.

Table illustrates a revised table design that stores the digest, rather than the plain text password. To make this table work, consider the code in Listing -, which updates the code from Listing 1 by adding a call to MD5 in the query. Calling MD5 can be done in either the SQL query or in PHP.

```
MD5("password");          // 5f4dcc3b5aa765d61d8327deb882cf99
```

UserID (int)	Username (varchar)	Password (varchar)
1	ricardo	5f4dcc3b5aa765d61d8327deb882cf99
2	randy	5f4dcc3b5aa765d61d8327deb882cf99

TABLE Users Table with MD5 Hash Applied to Password Field

```
//Insert the user with the password being hashed by MD5 first.
function insertUser($username,$password){
 $link = mysqli_connect("localhost", "my_user", "my_password",
                        "Login");
 $sql = "INSERT INTO Users(Username,Password)
         VALUES('$username',MD5('$password'))";
 mysqli_query($link, $sql); //execute the query
}
```

```php
//Check if the credentials match a user in the system with MD5 hash
function validateUser($username,$password){
 $link = mysqli_connect("localhost", "my_user", "my_password",
                        "Login");
 $sql = "SELECT UserID FROM Users WHERE Username='$username' AND
         Password=MD5('$password')";

 $result = mysqli_query($link, $sql);    //execute the query
 if($row = mysqli_fetch_assoc($result)){
   return true;  //record found, return true.
 }
 return false;  //record not found matching credentials, return false
}
```

LISTING PHP functions to insert and select a record using password hashing

```php
//Insert the user with the password
function insertUser($username,$password){
  $link = mysqli_connect("localhost", "my_user", "my_password",
                         "Login");
  $sql = "INSERT INTO Users(Username,Password)
          VALUES('$username','$password')");
  mysqli_query($link, $sql);            //execute the query
}

//Check if the credentials match a user in the system
function validateUser($username,$password){
  $link = mysqli_connect("localhost", "my_user", "my_password",
                         "Login");
  $sql = "SELECT UserID FROM Users WHERE Username='$username' AND
          Password='$password'";
  $result = mysqli_query($link, $sql);   //execute the query
  if($row = mysqli_fetch_assoc($result)){
    return true;   //record found, return true.
  }

  return false; //record not found matching credentials, return false
}
```

LISTING 1 PHP functions to insert and select a record with plaintext storage

Salting the Hash

A simple Google search for the string stored in our newly defined table: `5f4dcc3b5aa765d61d8327deb882cf99` brings up dozens of results which tell you that that string is indeed the MD5 digest for *password*. Although most hashes do not so easily appear in search engine results, many common ones do.

It turns out that a hacker with access to a table of hashes could build data structures called *rainbow tables* that aid in breaking passwords given enough time and space. However, if you add some unique *noise* to each digest, you prevent rainbow tables from defining the entire lookup space in one go. That is, the hacker would need to build a complete set of tables for each noisy password, making it practically impossible given current knowledge and computational power.

The technique of adding some noise to each password is called **salting** the password and makes your passwords very secure. The Unix system time can be used, or

UserID (int)	Username (varchar)	Password (varchar)	Salt
1	ricardo	edee24c1f2f1a1fda2375828fbeb6933	12345a
2	randy	ffc7764973435b9a2222a49d488c68e4	54321a

TABLE Users Table with MD5 Hash Using a Unique Salt in the Password Field

another pseudo-random string so that even if two users have the same password they have different digests, and are harder to decipher. Table shows an example of the correct way to store credentials, with passwords salted and encrypted with a one-way hash. In this example the passwords for randy and ricardo are still the same, but since they are hashed with different salts, it is not obvious that these two users have the same password. That is:

```
MD5("password12345a");      // edee24c1f2f1a1fda2375828fbeb6933
MD5("password54321a");      // ffc7764973435b9a2222a49d488c68e4
```

To make salting work, the code in Listing [illegible] makes use of a function to generate random strings when creating a new user. To authenticate, the code makes two queries, one to retrieve the salt and another to see if the login was correct by

```
function generateRandomSalt(){
 return base64_encode(mcrypt_create_iv(12), MCRYPT_DEV_URANDOM));
}
//Insert the user with the password salt generated, stored, and
//password hashed
function insertUser($username,$password){
 $link = mysqli_connect("localhost", "my_user", "my_password",
                        "Login");
 $salt = generateRandomSalt();
 $sql = "INSERT INTO Users(Username,Password,Salt)
         VALUES('$username',MD5('$password$salt'), '$salt')");
  mysqli_query($link, $sql); //execute the query
}

//Check if the credentials match a user in the system with MD5 hash
//using salt
function validateUser($username,$password){
 $link = mysqli_connect("localhost", "my_user", "my_password",
                        "Login");
 $sql = "SELECT Salt FROM Users WHERE Username='$username'";
 $result = mysqli_query($link, $sql);    //execute the query
 if($row = mysqli_fetch_assoc($result)){
```

```
    //username exists, build second query with salt
    $salt = $row['Salt'];
    $saltSql = "SELECT UserID FROM Users WHERE Username='$username'
                AND Password=MD5('$password$salt')";";
    $finalResult = mysqli_query($link, $saltSql);
    if($finalrow = mysqli_fetch_assoc($finalResult))
       return true;  //record found, return true.
  }
  return false; //record not found matching credentials, return false
}
```

LISTING PHP functions to insert and select a record using password hashing and salting

hashing the submitted value with the stored salt. As an exercise, try to improve this code by combining those two queries and logic into a single SQL query.

If you apply these principles to your systems, then you will immediately mitigate the impact of a successful attack that may happen in the future. While a hacker could still employ a brute force search to guess the passwords, this requires an investment of incredible computational power, which the hacker may not be prepared to commit to.

Note, at the time of writing this book, neither salted hash in our example appears in Google search results, further evidence that it's difficult to obtain the MD5 hash for every possible password, especially uncommon ones (which a salted password is likely to be).

Monitor Your Systems

You must see by now that breaches are inevitable. One of the best ways to mitigate damage is to detect an attack as quickly as possible, rather than let an attacker take their time in exploiting your system once inside. We can detect intrusion directly by watching login attempts, and indirectly by watching for suspicious behavior like a web server going down.

System Monitors

Now while you could periodically check your sites and servers manually to ensure they are up, it is essential to automate these tasks. There are tools that allow you to pre-configure a system to check in on all your sites and servers periodically. Nagios, for example, comes with a web interface as shown in Figure that allows you to see the status and history of your devices, and sends out notifications by email as per your preferences. There is even a marketplace to allow people to buy and sell plug-ins that extend the base functionality.

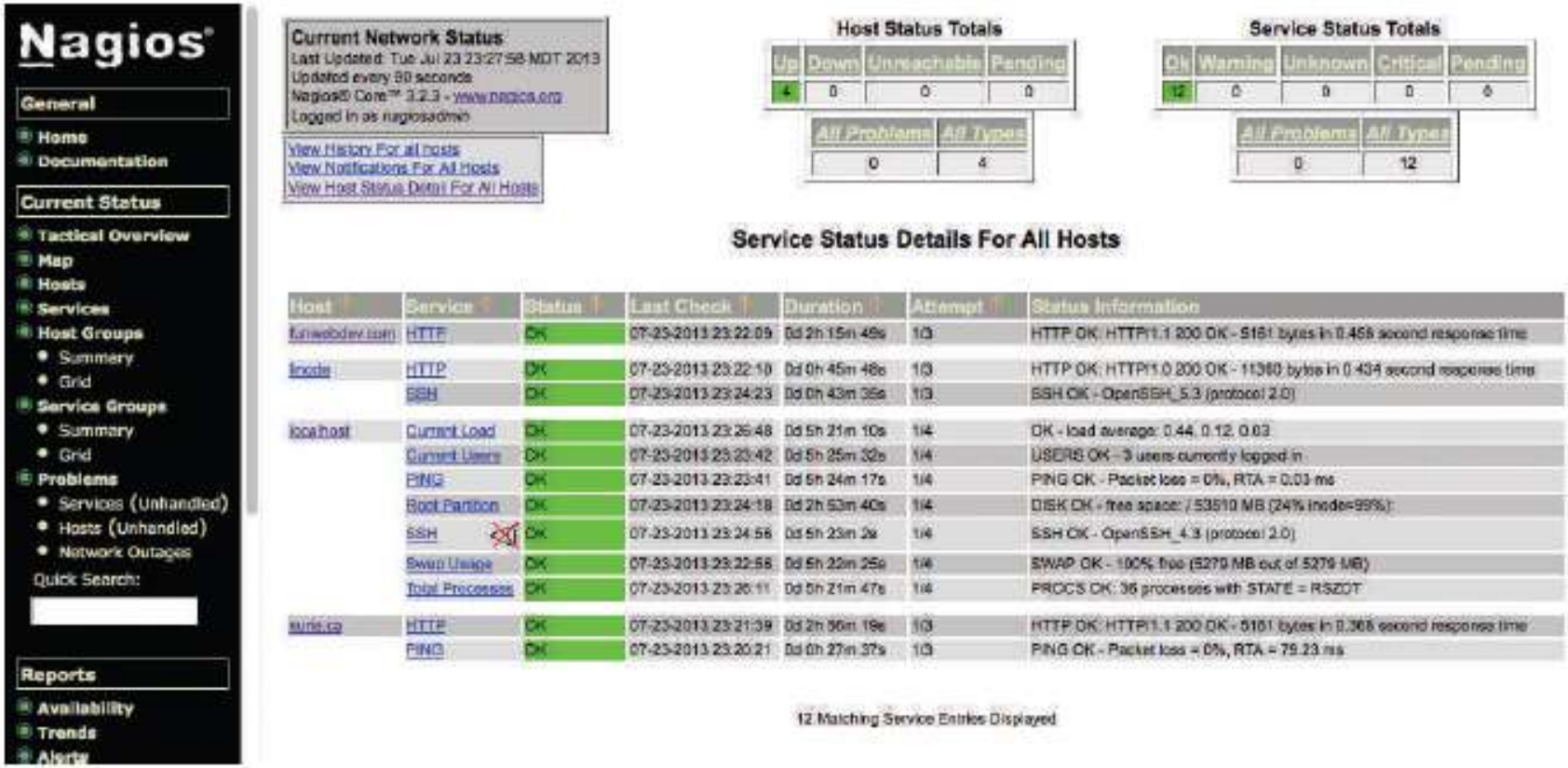


FIGURE Screenshot of the Nagios web interface (green means OK)

Access Monitors

As any experienced site administrator will attest, there are thousands of attempted login attempts being performed all day long, mostly from Eurasian IP addresses. They can be found by reading the log files often stored in /var/log/. Inside those files attempted login attempts can be seen as in Listing .

Inside of the /var/log directory there will be multiple files associated with multiple services. Often there is a mysql.log file for MySQL logging, access_log file for HTTP requests, error_log for HTTP errors, and secure for SSH connections. Reading these files is normally permitted only to the root user to ensure no one else can change the audit trail that is the logs.

If you did identify an IP address you wanted to block (from SSH for example), you could add the address to etc/hosts.deny (or hosts.allow with a deny flag).

```
Jul 23 23:35:04 funwebdev sshd[19595]: Invalid user randy from
    68.182.20.18
Jul 23 23:35:04 funwebdev sshd[19596]: Failed password for invalid
    user randy from 68.182.20.18 port 34741 ssh2
```

LISTING Sample output from a secure log file showing a failed SSH login

Addresses in hosts.deny are immediately prevented from accessing your server. Unfortunately, hackers are attacking all day and night, making this an impossible activity to do manually. By the time you wake up several million login attempts could have happened.

Automated Intrusion Blocking

Automating intrusion detection can be done in several ways. You could write your own PHP script that reads the log files and detects failed login attempts, then uses a history to determine the originating IP addresses to automatically add to hosts.deny. This script could then be run every minute using a cron job (scheduled task) to ensure round-the-clock vigilance.

For those of us less interested in writing that script from scratch, consider the well-tested and widely used Python script blockhosts.py or other similar tools like failzban or blockhostz. These tools look for failed login attempts by both SSH and FTP and automatically update hosts.deny files as needed. You can configure how many failed attempts are allowed before an IP address is automatically blocked and create your own custom filters.[20]

Audit and Attack Thyself

Attacking the systems you own or are authorized to attack in order to find vulnerabilities is a great way to detect holes in your system and patch them before someone else does. It should be part of all the aspects of testing, including the deployment tests, but also unit testing done by developers. This way SQL injection, for example, is automatically performed with each unit test, and vulnerabilities are immediately found and fixed.

There are a number of companies that you can hire (and grant written permission) to test your servers and report on what they've found. If you prefer to perform your own analysis, you should be aware of some open-source attack tools such as *w3af*, which provide a framework to test your system including SQL injections, XSS, bad credentials, and more.[21] Such a tool will automate many of the most common types of attack and provide a report of the vulnerabilities it has identified.

With a list of vulnerabilities, reflect on the risk assessment (not all risks are worth addressing) to determine which vulnerabilities are worth fixing.

Common Threat Vectors

A badly developed web application can open up many attack vectors. No matter the security in place, there are often backdoors and poorly secured resources, which are accidentally left accessible to the public. This section describes some common attacks and some countermeasures you can apply to mitigate their impact.

SQL Injection

SQL injection is the attack technique of using reserved SQL symbols to try and make the web server execute a malicious query other than what was intended. This vulnerability is an especially common one because it targets the programmatic construction of SQL queries, which, as we have seen, is an especially common feature of most database-driven websites.

Consider a vulnerable application illustrated in Figure .

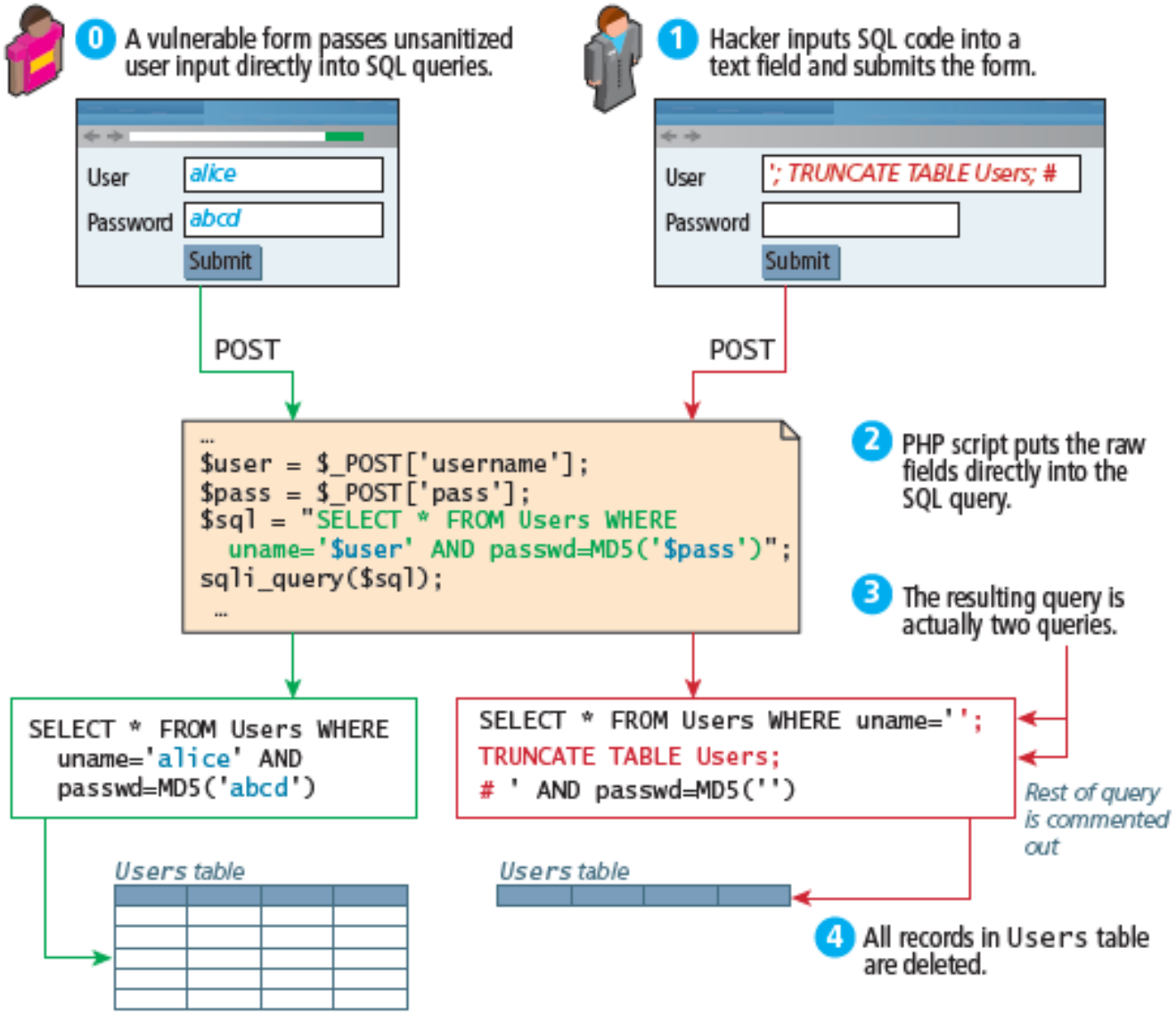


FIGURE Illustration of a SQL injection attack (right) and intended usage (left)

In this web page's intended-usage scenario (which does work), a username and a password are passed directly to a SQL query, which will either return a result (valid login) or nothing (invalid). The problem is that by passing the user input directly to the SQL query, the application is open to SQL injection. To illustrate, in Figure 1 the attacker inputs text that resembles a SQL query in the username field of the web form. The malicious attacker is not trying to log in, but rather, trying to insert rogue SQL statements to be executed that have nothing to do with the user authentication system. Once submitted to the server, the user input actually results in two distinct queries being executed:

```
1. SELECT * FROM Users WHERE uname='';
2. TRUNCATE TABLE User;
```

The second one (`TRUNCATE`) removes all the records from the `Users` table, effectively wiping out all the user records, making the site inaccessible to all registered users!

Try to imagine what kind of damage hackers could do with this technique since they are only limited by the SQL language, the permission of the database user, and their ability to decipher the table names and structure. While we've illustrated an attack to break a website (availability attack), it could just as easily steal data (confidentiality attack) or insert bad data (integrity attack), making it a truly versatile technique.

There are two ways to protect against such attacks: sanitize user input, and apply the least privileges possible for the application's database user.

Sanitize Input

To sanitize user input (remember, user input is often achieved through query strings) before using it in a SQL query, you either apply sanitization functions (`mysqli_real_escape_string`) or bind the variables in the query using parameters or prepared statements.

From a security perspective, you should never trust a user input enough to use it directly in a query, no matter how many HTML5 or JavaScript pre-validation techniques you use. Remember that at the end of the day your server responds to HTTP requests, and a hacker could easily circumvent your JavaScript and HTML5 prevalidation and post directly to your server.

Least Possible Privileges

Despite the sanitization of user input, there is always a risk that users could somehow execute a SQL query they are not entitled to. A properly secured system only assigns users and applications the privileges they need to complete their work, but

For instance, in a typical web application, one could define three types of database user for that web application: one with read-only privileges, one with write privileges, and finally an administrator with the ability to add, drop, and truncate tables. The read-only user is used with all queries by nonauthenticated users. The other two users are used for authenticated users and privileged users, respectively.

In such a situation, the SQL injection example would not have worked, even if the query executed since the read-only account does not have the `TRUNCATE` privilege and therefore the attack does not work.

Cross-Site Scripting (XSS)

Cross-site scripting (called XSS, so as not to be confused with CSS) refers to a type of attack in which a malicious script (JavaScript, VBScript, or ActionScript) is embedded into an otherwise trustworthy website. These scripts can cause a wide range of damage and can do just about anything you as developers could do writing a script on your own page.

In the original formulation for these type of attacks, a malicious user would get a script onto a page and that script would then send data through AJAX to a malicious party, hosted at another domain (hence the cross, in XSS). That problem has been partially addressed by modern browsers, which restrict AJAX requests to the same domain. However, with at least 80 XSS attack vectors to get around those restrictions, it remains a serious problem.[22] There are two main categories of XSS vulnerability: **Reflected XSS** and **Stored XSS**. They both apply similar techniques, but are distinct attack vectors.

Reflected XSS

Reflected XSS (also known as nonpersistent XSS) are attacks that send malicious content to the server, so that in the server response, the malicious content is embedded.

For the sake of simplicity, consider a login page that outputs a welcome message to the user, based on a `GET` parameter. For the URL index.php?User=eve, the page might output `Welcome eve!` as shown in Figure 1.

A malicious user could try to put JavaScript into the page by typing the URL:

```
index.php?User=<script>alert("bad");<script>
```

What is the goal behind such an attack? The malicious user is trying to discover if the site is vulnerable, so they can craft a more complex script to do more damage. For instance, the attacker could send known users of the site an email including a link containing the JavaScript payload, so that users that click the link will be exposed to a version of the site with the XSS script embedded inside as illustrated in Figure 4. Since the domain is correct, they may even be logged in automatically, and start transmitting personal data (including, for instance, cookie data) to the malicious party.

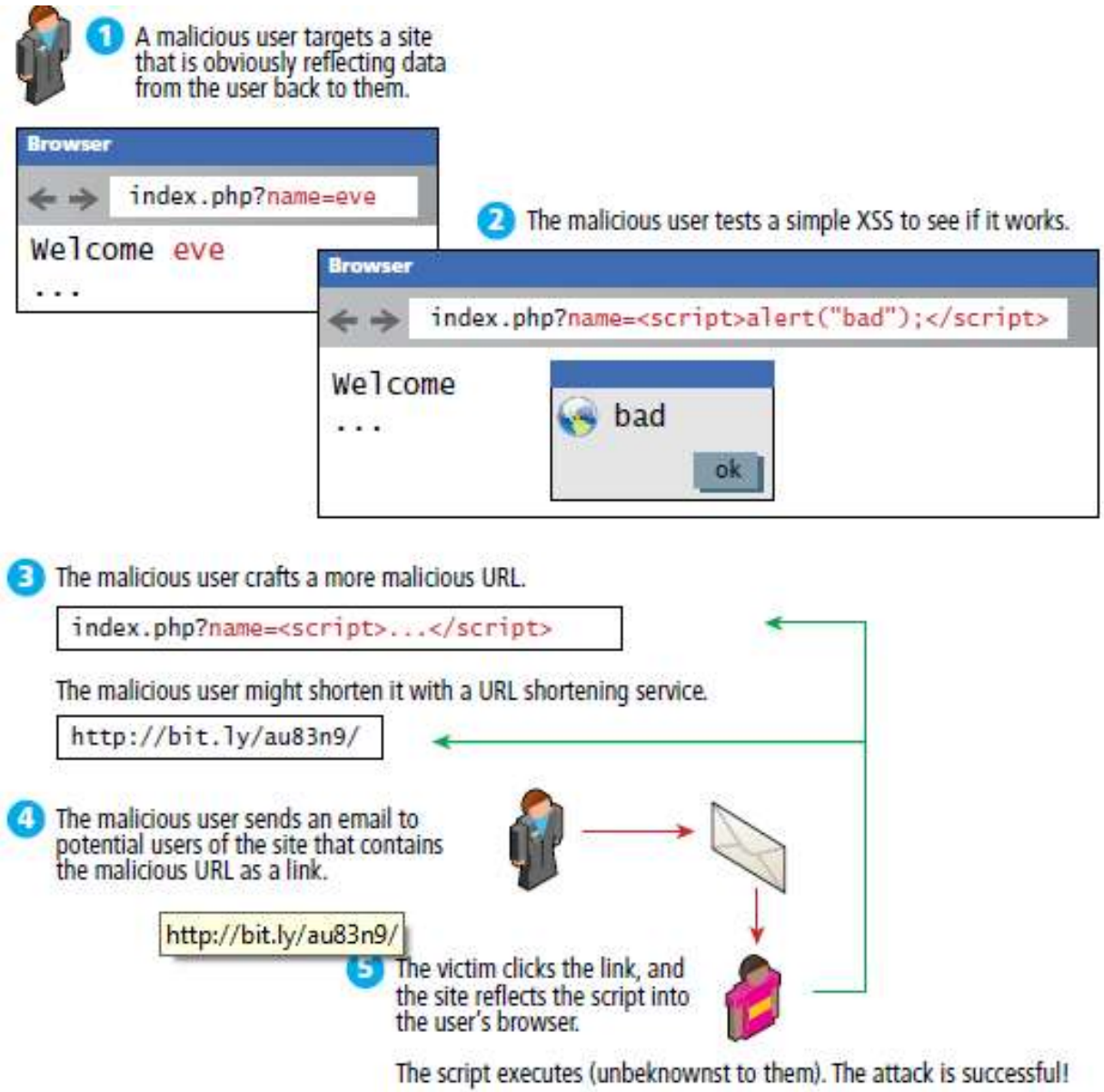


FIGURE Illustration of a Reflection XSS attack

Stored XSS

Stored XSS (also known as persistent XSS) is even more dangerous, because the attack can impact every user that visits the site. After the attack is installed, it is transmitted to clients as part of the response to their HTTP requests. These attacks are embedded into the content of a website (in one's database) and can persist forever or until detected!

To illustrate the problem, consider a blogging site, where users can add comments to existing blog posts. A malicious user could enter a comment that includes malicious JavaScript, as shown in Figure . Since comments are saved to the database, the script is now embedded into the web page. The next time the administrator logs in (actually every time anyone logs in), their session cookie will be transmitted to the malicious site as an innocent-looking image request. The malicious user can now use that secret session value in their server logs and gain access to the site as though they were an administrator simply by using that cookie with a browser plug-in that allows cookie modification.

As you can see XSS relies extensively on un-sanitized user inputs to operate; preventing XSS attacks, therefore, requires even more user input sanitization, just as SQL injection defenses did.

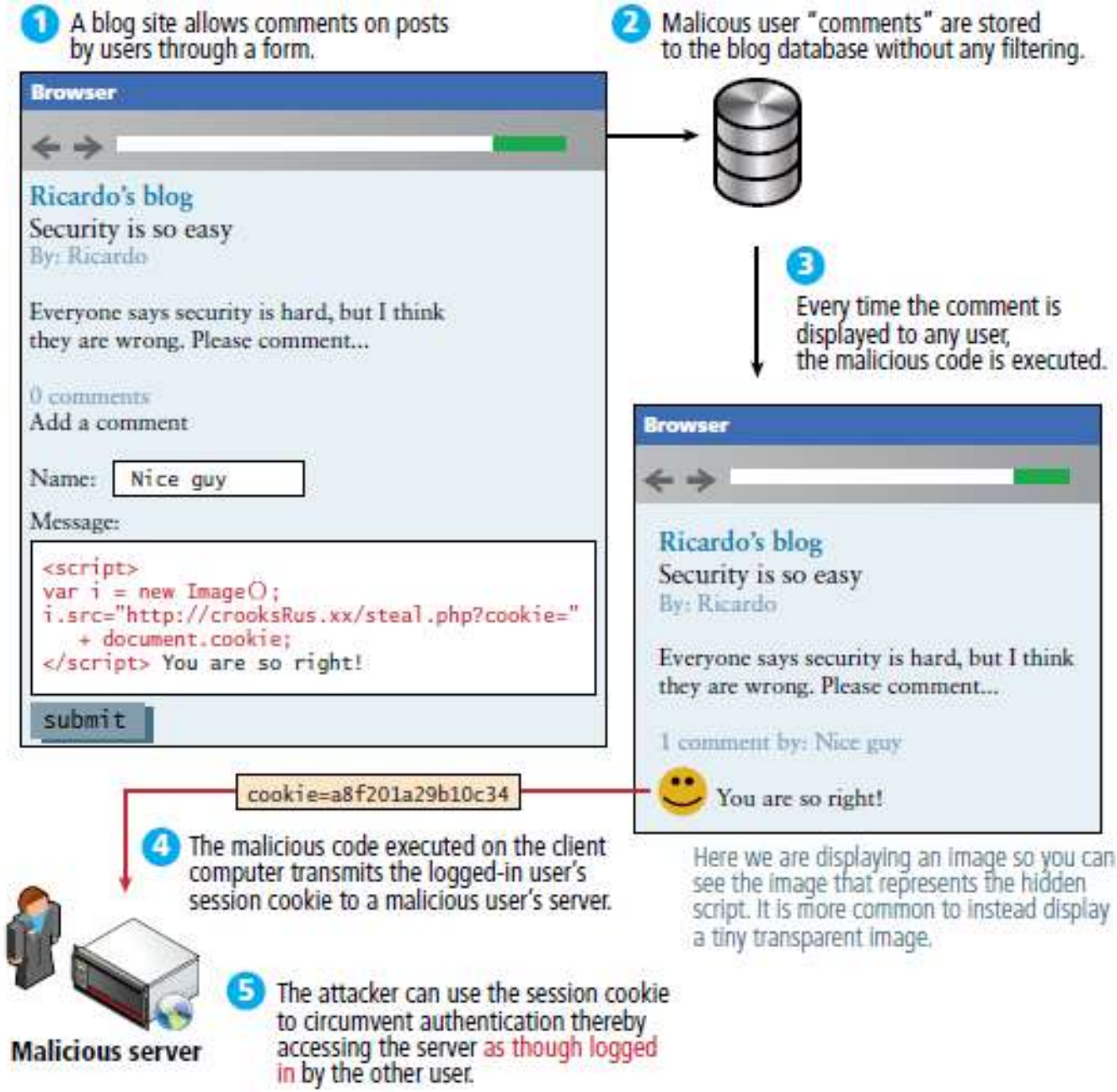


FIGURE Illustration of a stored XSS attack in action

Filtering User Input

Obviously sanitizing user input is crucial to preventing XSS attacks, but as you will see filtering out dangerous characters is a tricky matter. It's rather easy to write PHP sanitization scripts to strip out dangerous HTML tags like `<script>`. For example, the PHP function `strip_tags()` removes all the HTML tags from the passed-in string. Although passing the user input through such a function prevents the simple script attack, attackers have gone far beyond using HTML script tags, and commonly employ subtle tactics including embedded attributes and character encoding.

- Embedded attributes use the attribute of a tag, rather than a `<script>` block, for instance:

```
<a onmouseover="alert(document.cookie)">some link text</a>
```

- Hexadecimal/HTML encoding embeds an escaped set of characters such as:

```
%3C%73%63%72%69%70%74%3E%61%6C%65%72%74%28%22%68%65%6C%6C%6F%22%29%3
B%3C%2F%73%63%72%69%70%74%3E
```

instead of `<script>alert("hello");</script>`.

This technique actually has many forms including hexadecimal codes, HTML entities, and UTF-8 codes.

Given that there are at least 80 subtle variations of those types of filter evasions, most developers rely on third-party filters to remove dangerous scripts rather than develop their own from scratch. A library such as the open-source HTMLPurifier from http://htmlpurifier.org/ or HTML sanitizer from Google[23] allows you to easily remove a wide range of dangerous characters from user input that could be used as part of an XSS attack. Using the downloadable HTMLPurifier.php, you can replace the usage of `strip_tags()` with the more advanced purifier, as follows:

```
$user= $_POST['uname'];
$purifier = new HTMLPurifier();
$clean_user = $purifier->purify($user);
```

Escape Dangerous Content

Once content is in the database, there are still techniques to prevent an attack from being successful. Escaping content is a great way to make sure that user content is never executed, even if a malicious script was uploaded. This technique relies on the fact that browsers don't execute escaped content as JavaScript, but rather interpret it as text. Ironically, it uses one of the techniques the hackers employ to get past filters.

You may recall HTML escape codes allow characters to be encoded as a code, preceded by &, and ending with a semicolon (e.g., < can be encoded as `<`). That means even if the malicious script did get stored, you would escape it before sending it out to users, so they would receive the following:

```
&lt;script&gt;alert(&quot;hello&quot;);&lt;/script&gt;
```

The browsers seeing the encoded characters would translate them back for display, but will not execute the script! Instead your code would appear on the page as text. The Enterprise Security API (ESAPI), maintained by the Open Web Application Security Project, is a library that can be used in PHP, ASP, JAVA, and many other server languages to escape dangerous content in HTML, CSS, and JavaScript[24] for more than just HTML codes.

The trick is not to escape everything, or your own scripts will be disabled! Only escape output that originated as user input since that could be a potential XSS attack vector (normally, that's the content pulled from the database). Combined with user input filtering, you should be well prepared for the most common, well-known XSS attacks.

XSS is a rapidly changing area, with HTML5 implementations providing even more potential attack vectors. What works today will not work forever, meaning this threat is an ongoing one.

Insecure Direct Object Reference

An insecure direct object reference is a fancy name for when some internal value or key of the application is exposed to the user, and attackers can then manipulate these internal keys to gain access to things they should not have access to.

One of the most common ways that data can be exposed is if a configuration file or other sensitive piece of data is left out in the open for anyone to download (that is, for anyone who knows the URL). This could be an archive of the site's PHP code or a password text file that is left on the web server in a location where it could potentially be downloaded or accessed.

Another common example is when a website uses a database key in the URLs that are visible to users. A malicious (or curious) user takes a valid URL they have access to and modifies it to try and access something they do not have access to. For instance, consider the situation in which a customer with an ID of 99 is able to see his or her profile page at the following URL: `info.php?CustomerID=99`. In such a site, other users should not be able to change the query string to a different value (say, 100) and get the page belonging to a different user (i.e, the one with ID 100). Unfortunately, unless security authorization is checked with each request for a resource, this type of negligent programming leaves your data exposed.

Another example of this security risk occurs due to a common technique for storing files on the server. For instance, if a user can determine that his or her uploaded photos are stored sequentially as /images/99/1.jpg, /images/99/2.jpg, . . . , they might try to access images of other users by requesting /images/101/1.jpg.

One strategy for protecting your site against this threat is to obfuscate URLs to use hash values rather than sequential names. That is, rather than store images as 1.jpg, 2.jpg . . . use a one-way hash, so that each user's images are stored with unique URLs like 9a76eb01c5de4362098.jpg. However, even obfuscation leaves the files at risk for someone with enough time to seek them by brute force.

If image security is truly important, then image requests should be routed through PHP scripts rather than link to images directly. This is one significant advantage of linking to scripts that use BLOB storage in your database rather than files, since the PHP script already serves the images and therefore we can easily add an authorization check for every picture using the `$_SESSION` variable.

Denial of Service

Denial of service attacks (DoS attacks) are attacks that aim to overload a server with illegitimate requests in order to prevent the site from responding to legitimate ones.

If the attack originates from a single server, then stopping it is as simple as blocking the IP address, either in the firewall or the Apache server. However, more recently these attacks have become distributed, making them harder to protect against as shown in Figure .

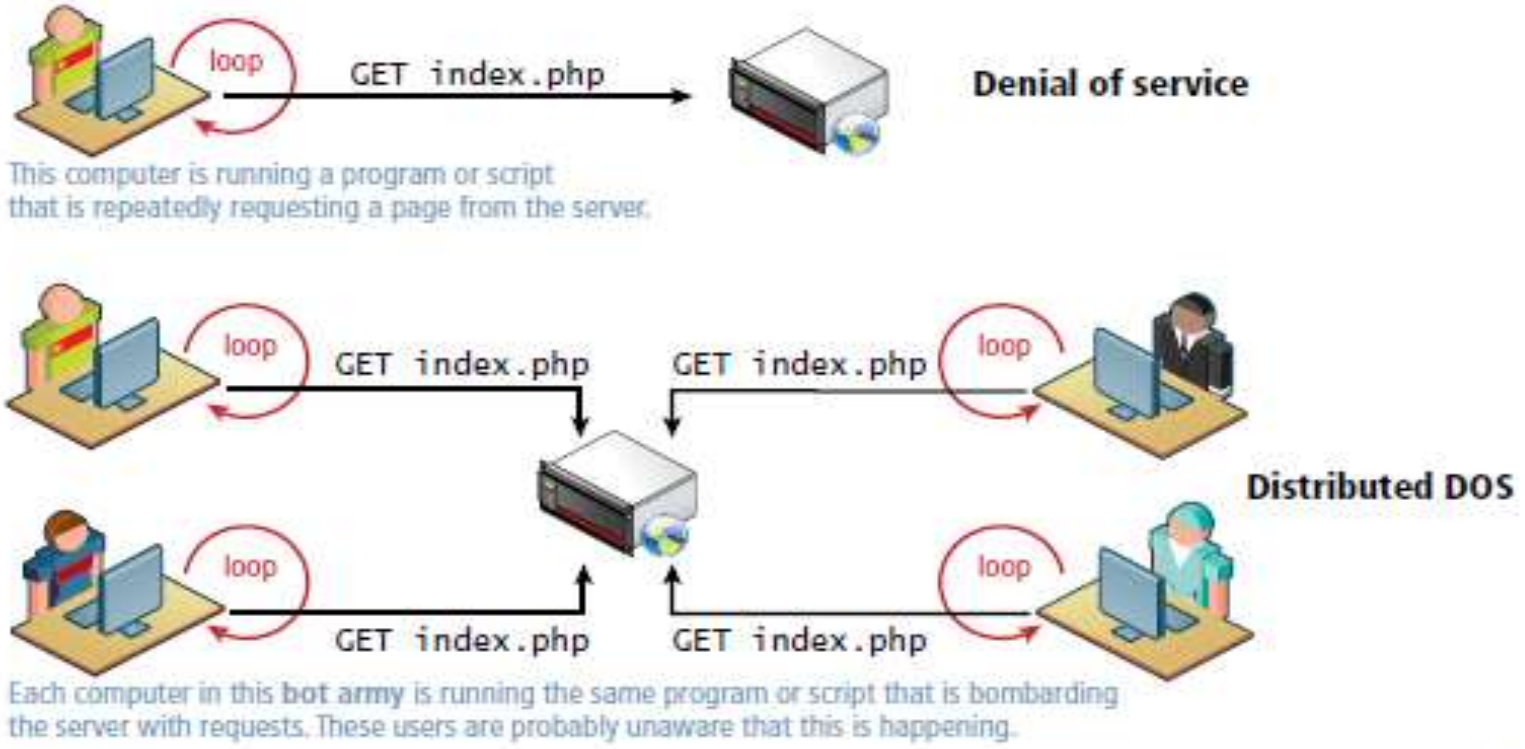


FIGURE : Illustration of a Denial of Service (DoS) and a Distributed Denial of Service (DDoS) attack

Distributed DoS Attack (DDoS)

The challenge of DDoS is that the requests are coming in from multiple machines, often as part of a bot army of infected machines under the control of a single organization or user. Such a scenario is often indistinguishable from a surge of legitimate traffic from being featured on a popular blog like reddit or slashdot.

Unlike a DoS attack, you cannot block the IP address of every machine making requests, since some of those requests are legitimate and it's difficult to distinguish between them.

Interestingly, defense against this type of attack is similar to preparation for a huge surge of traffic, that is, caching dynamic pages whenever possible, and ensuring you have the bandwidth needed to respond. Unfortunately, these attacks are very difficult to counter, as illustrated by a recent attack on the spamhaus servers, which generated 300Gbps worth of requests![25]

Security Misconfiguration

The broad category of security misconfiguration captures the wide range of errors that can arise from an improperly configured server. There are more issues that fall into this category than the rest, but some common errors include out-of-date software, open mail relays, and user-coupled control.

Out-of-Date Software

Most softwares are regularly updated with new versions that add features, and fix bugs. Sometimes these updates are not applied, either out of laziness/incompetence, or because they conflict with other software that is running on the system that is not compatible with the new version.

From the OS and services, all the way to updates for your plug-ins in Wordpress, out-of-date software puts your system at risk by potentially leaving well-known (and fixed) vulnerabilities exposed.

The solution is straightforward: update your software as quickly as possible. The best practice is to have identical mirror images of the production system in a preproduction setting. Test all updates on that system before updating the live server.

Open Mail Relays

An open mail relay refers to any mail server that allows someone to route email through without authentication. Open relays are troublesome since spammers can use your server to send their messages rather than use their own servers. This means that the spam messages are sent as if the originating IP address was your own web server! If that spam is flagged at a spam agency like spamhaus, your mail server's IP address will be blacklisted, and then many mail providers will block legitimate email from you.

A proper closed email server configuration will allow sending from a locally trusted computer (like your web server) and authenticated external users. Even when properly configured from an SMTP (Simple Mail Transfer Protocol) perspective, there can still be a risk of spammers abusing your server if your forms are not correctly designed, since they can piggyback on the web server's permission to route email and send their own messages.

More Input Attacks

Although SQL injection is one type of unsanitized user input that could put your site at risk, there are other risks to allowing user input to control systems. **Input coupled control** refers to the potential vulnerability that occurs when the users, through their HTTP requests, transmit a variety of strings and data that are directly used by the server without sanitation. Two examples you will learn about are the virtual open mail relay and arbitrary program execution

Virtual Open Mail Relay

Consider, for example, that most websites use an HTML form to allow users to contact the website administrator or other users. If the form allows users to select the recipient from a dropdown, then what is being transmitted is crucial since it could expose your mail server as a virtual open mail relay as illustrated in Figure .

By transmitting the email address of the recipient, the contact form is at risk of abuse since an attacker could send to any email they want. Instead, you should transmit an integer that corresponds to an ID in the user table, thereby requiring the database lookup of a valid recipient.

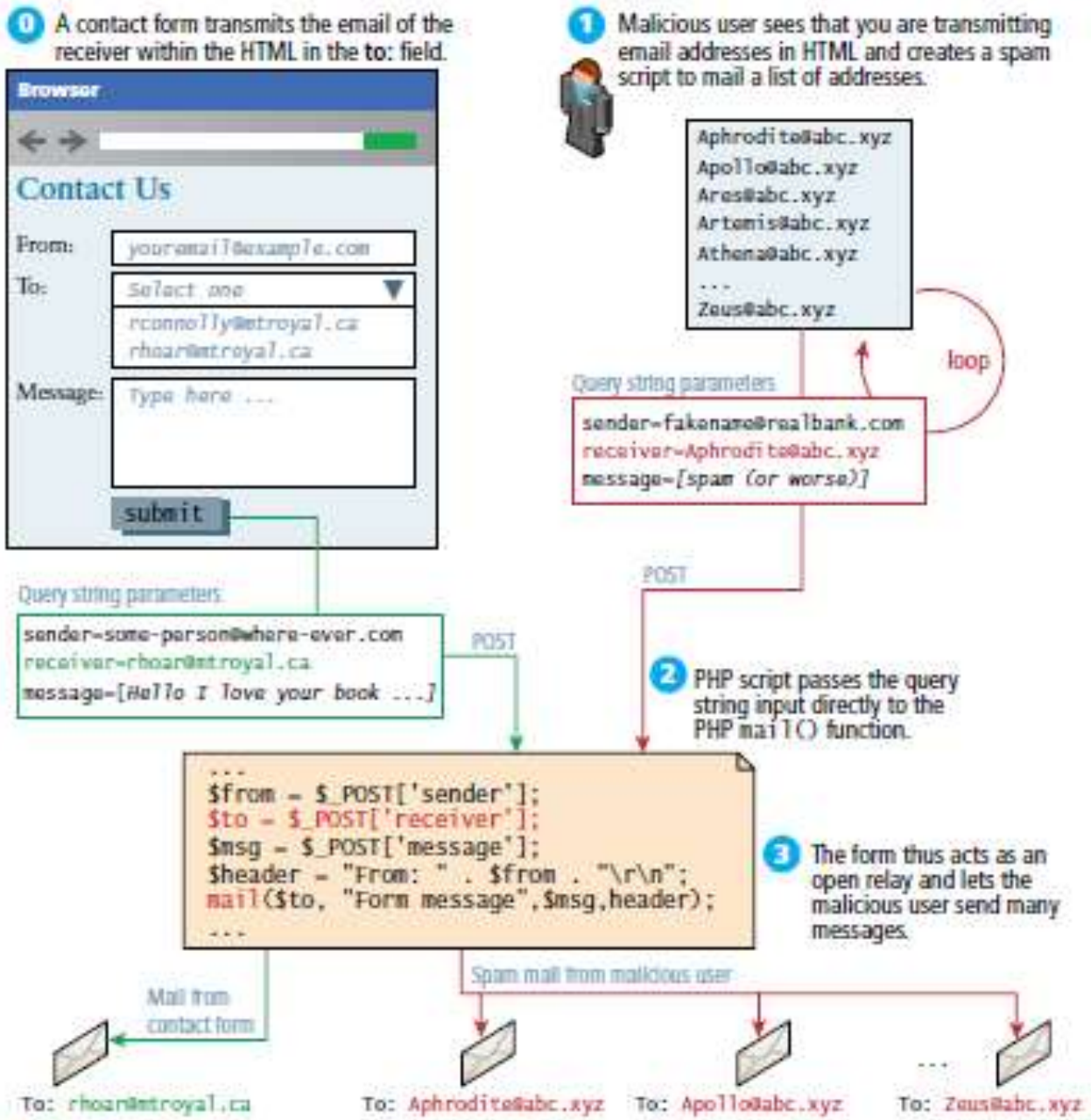


FIGURE Illustrated virtual open relay exploit

Arbitrary program execution

Another potential attack with user-coupled control relates to running commands in Unix through a PHP script. Functions like `exec()`, `system()`, and `passthru()` allow the server to run a process as though they were a logged-in user.

Consider the script illustrated in Figure , which allows a user to input an IP address (or domain name) and then runs the `ping` command on the server using that input. Unfortunately, a malicious user could input data other than an IP address in an effort to break out of the `ping` command and execute another command. These attackers normally use | or > characters to execute the malicious program as part of a chain of commands. In this case the attacker appends a directory listing command (`ls`), and as a result sees all the files on the server in that directory! With access to any command, the impact could be much worse. To prevent this major class of attack, be sure to sanitize input, with `escapeshellarg()` and be mindful of how user input is being passed to the shell.

Applying least possible privileges will also help mitigate this attack. That is, if your web server is running as root, you are potentially allowing arbitrary commands to be run as root, versus running as the Apache user, which has fewer privileges.

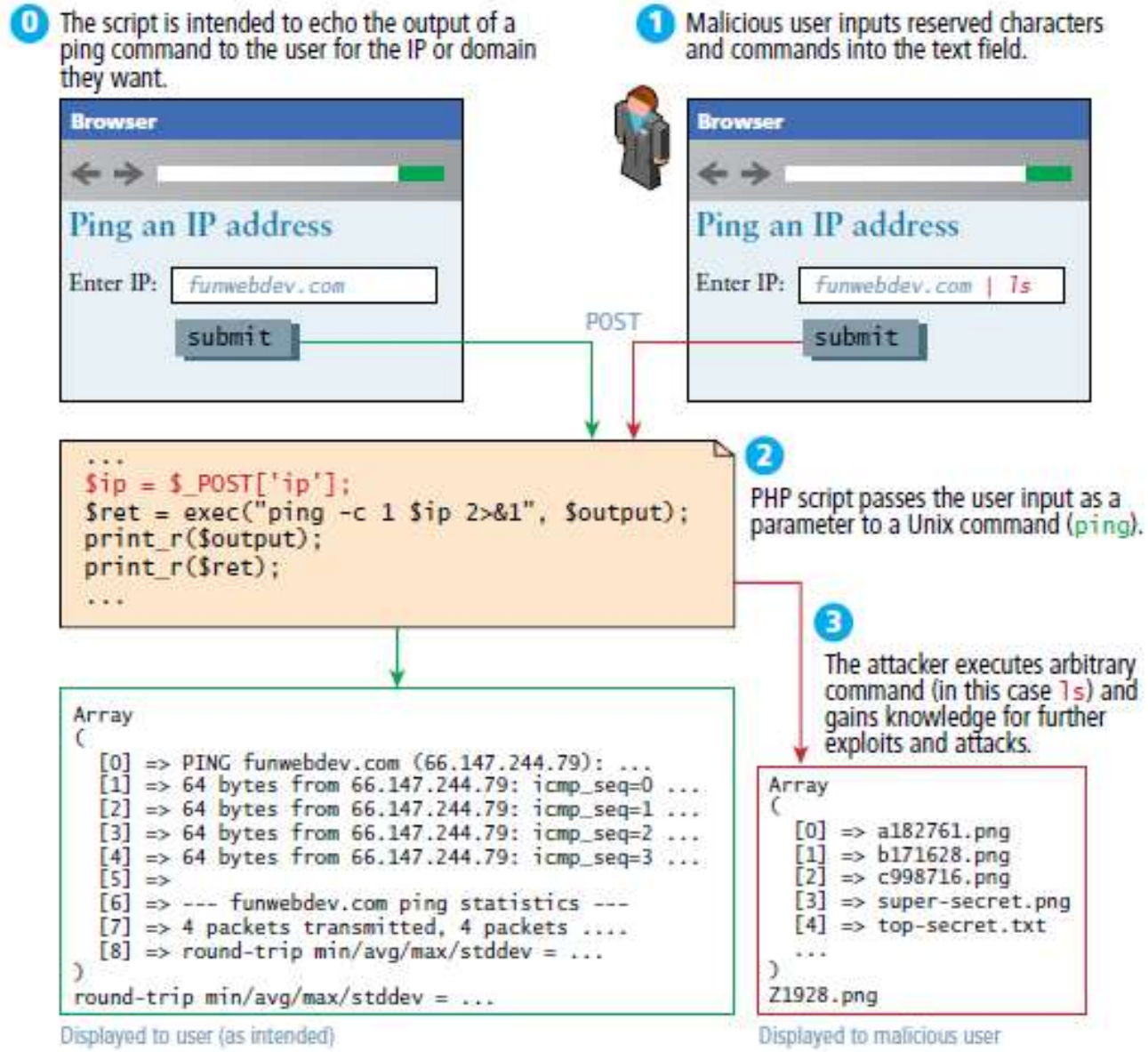


FIGURE Illustrated exploit of a command-line pass-through of user input

Review Questions

1. What are the three components of the CIA security triad?
2. What is the difference between authentication and authorization?
3. Why is two-factor authentication more secure than single factor?
4. How does the *secure by design* principle get applied in the software development life cycle?
5. What are the three types of actor that could compromise a system?
6. What is security theater? Is it effective?
7. What's the relationship between the Caesar cipher and the modern RSA cipher?
8. What type of cryptography addresses the problem of agreeing to a secret symmetric key?
9. What is a cryptographic one-way hash?
10. What does it mean to salt your passwords?
11. What is a Certificate Authority, and why do they matter?
12. What is a DOS attack, and how does it differ from a DDOS attack?
13. What can you do to prevent SQL injection vulnerabilities?
14. What's the difference between reflected and stored XSS attacks?
15. How do you defend against cross-site scripting attacks?
16. What features does a digital signature provide?
17. What is a self-signed certificate?