JOHANNES KEPLER
UNIVERSITY LINZ | JKU

Introduction to Web Security

Michael Sonntag

Institute for Information processing and microprocessor technology (FIM)

Johannes Kepler University Linz, Austria

sonntag@fim.uni-linz.ac.at

Why attack web applications/servers?

- Ubiquity: Every company has a web server, and many a shop, Web 2.0 elements etc. → If not here, try the next million sites!
- Simple techniques: Protocols are simple and attacks are widely known
  - Text input → Manipulation is easy and many tools exist/custom programming trivial
  - OS buffer overflow: Compare this to SQL injection (finding & exploiting)!
- Anonymity: You don't need to be there; proxies for HTTP(S) are everywhere
- Firewall bypassing: Web traffic is always allowed in both directions: in & out
- Custom code: Web application programming is simple, so "simpletons" do it!
  - No security education for them!
- Immature security: No sessions, authentication weak: "Do everything yourself"
- Constant change: Numerous persons always change the web application
  - OS: Producer, Software: Rare updates only

What can go wrong?

- Denial of Service (DoS): Your webshop is not accessible → Direct losses
  - Party/company website down: Reputation loss, …
- Defacement: The content of the website is changed
  - Shop: Price modifications for you or all customers → Expensive!
  - Reputation loss, shutdown by government (e.g. illegal content), added to blacklists etc.
- Data loss: Data from you (or your customers/employees/…!) is stolen
  - This is usually no immediate problem – you still have it
  - But the consequences can be dire: Trade secrets lost, fines/compensations to pay, bad reputation/customer loss, …
- Service stealing: Whatever service/data you provide is used for free
- "Piggybacking": Using your resources, e.g. to send Spam, host own data, phishing, infect visitors, …
  - Liability, increased costs, lower performance, blacklists etc.

Where to attack?

- Operating System: Not covered here; remote attacks rare and difficult
- Transport: HTTP / TLS sniffing; extremely difficult if not on path
- Web Server: The server itself and any necessary applications/languages
  - PHP, Python, Ruby, …/Server plugins
- Web application platform: Basic frameworks used by the application
  - Spring, JSF, Ruby on Rails, Struts, Typestry, Cold Spring,…(Drupal, Typo3…)
- Database: Typically only an indirect target
  - PostgreSQL, MySQL, MS SQL Server, DB2, Oracle; any non-relational ones
- Web application: Server-side → See later!
- Web Client: Client-side → See later!
- Availability: Sending enormous amount of requests (few variations)
  - Any kind of packets or full legitimate connections

How to attack: Profiling

- Gathering information on potential vulnerabilities (or excluding non-working ones)
  - Basic information: What OS, web server, application framework, language, load balancers, local time&timezone, proxies, web application firewall, services running,…
  - User information: Who owns it? Names, E-Mail addressed, IP addresses (web server, DNS, internal servers, …)
  - Website information: Complete mirror of pages, known accounts, static/dynamic pages, form pages, directory layout, presence of common files, source code (accessible, open source repositories, …) etc.
  - Vulnerability information:
    - Common profiles for applications/frameworks
    - Automated scanners for testing known ones
    - Manually looking for potential problems
  - Gathering data on vulnerabilities: How to exploit it, working code, assembling payload, preparing server for further code/control, …

How to attack: Executing the attack

- When will the system be most likely un-/less supervised?
- Exploiting the vulnerability or testing for any probable vulnerability
  - Or just testing anything, perhaps we are lucky!
  - Repeating the test – some are not deterministic
- Hiding traces of the attack while in progress (logs)
- Hiding the source of the attack (IP)
- Injecting the first "foothold": Typically some (root/Administrator) shell
  - Almost always fragile: Only in memory, very small, almost no functionality
- Expanding the foothold: Connecting back out, loading additional code, privilege escalation, installing permanently …
  - Has often to be done blindly, e.g. by a pre-defined script!
- Gaining access: Installing a backdoor/command receiver and testing it

Hiding: Various traces (1)

- Source (IP): Use another computer as proxy
  - Commercial, hacked, …
  - Or third parties, e.g. through SPAM
    - Let them try; if the attack is successful, the hacked computer will "phone home" – but to you, not to the third party!
- Source: Don't mix legitimate and "attack" traffic
  - Logging in and then trying → Bad idea!
  - Use different IP addresses and no connection data (e.g. session IDs!)
  - Different systems and different times
  - Use different credentials
- Progress: Rare tries
  - Not a single barrage of requests, but one every few hours split over several days – The server is going nowhere!

Hiding: Various traces (2)

- Progress: Huge tries (try to fill up the log and crash the computer after the attack)
- Progress: Log evasion (staying out of the log files)
  - Long URLs: Some logs limit the size of log entries (to avoid DoS attacks!)
    - So add harmless parameters in front of the "real" ones (depends on server used)!
    - E.g. http://victim.com/sh_prod.asp?uid=<4096 random chars>&uname=' or 1=1; --
  - Encoding: Encode everything as URLEncode
    - Makes it harder to see the attack unless explicitly looking
    - E.g. uname%3D'%20or%201%3D1%3B%20--%20
    - Or: %75%6e%61%6d%64%3d%27%20%6f%72%20%31%3d%31%3b%20%2d%2d%20
- Progress: Evading IDS (Intrusion Detection Systems)
  - E.g. inserting packets into a stream, which are physically addressed at the IDS (MAC address only; IP is for attacking connection!): Sees different data stream than recipient
    - Especially the IP and TCP layers allow numerous "errors" to confuse listeners

How to attack: Exploiting

- Install some malware: Typically a hidden "remote control" application
- Depending on the intention, various avenues are open:
  - Political/personal gain: Deface the website, download for free etc.
  - "Terrorist": Delete data, crash system
  - Espionage: Steal specific confidential information
  - Crime: Steal any data which might be worth something
    - Credit card/identity numbers, account credentials, E-Mail addresses
    - Introduce slight modifications into data: Bank account for payments
- Note: Very often the actual user of the illegal access is someone else;
  - "Renting" computers (botnets)
  - Selling raw data for exploitation by others
  - Selling the software itself (without any hacked computers!)

Who is responsible for web security?

- More persons than you probably thought!
  - The developer: Writing the application so it is secure
    - Or at least: Can be configured/used in a secure way
  - The webserver operator: Do not add insecurity through configuration
    - And make sure the web server, framework, application is installed securely
  - The network operator(s): Prevent attacks on the infrastructure
    - E.g. DNS attacks are very dangerous!
  - The end user: Use up-to-date clients as well as common sense
- Why? Server + Transport + Client must be secure (all of them simultaneously!)
  - Server insecure: Others can modify data on it, …
  - Transport insecure: Eavesdropping, MitM, …
  - Client: Some attacks can only be prevented by the client (like phishing)!

What do you have to do?

- Generally for security and specifically for web security:
  - Authenticate users: Is the person really who he/she claims to be?
  - Authorize users: Restrict the users to what they are allowed to do
    - Accessing/modifying/deleting data (files, webpages, DB content, …)
    - Check content: Even if from you, you shouldn't distribute infected webpages/files
  - Prevent eavesdropping: Nobody else should be able to access information being transmitted (stored → authorization!)
  - Ensure availability: DoS attacks, resource exhaustion/overload etc.
    - Icluding technical problems, force majeure, …
  - Tracing: Ensure that enough logging exists to be able to identify the source of attacks or any undesirable behaviour of the system (might also be legally required)
- Practical 80/20 rule: Protect the 20% of the system, which are high-impact and high-risk areas first to get rid of 80% of all incidents!

How to do it: Web application guidelines

- Build security in from the start ("good enough security")
  - Especially take care of adding "hooks" to improve/add security later!
  - Investigate what are the main assets to protect and what are potential attackers
- Test security: Not only functional testing, but also for security
  - Input which is deliberately wrong or strange; don't bank on random/ape tests to find it!
- Keep it simple and centralized
  - E.g. one point where every request must pass through
  - Give out only as much information as necessary (esp. error messages; but: local logs)
- Store all data unescaped and raw → Escape all data when creating output (according to location) or using it (e.g. as commands, DB query content)
  - Because you don't know where it will end up, so you can't appropriately escape it!
- Don't do it yourself: Cryptography, authentication etc.
  - Should be part of framework used → Use it securely (and actually use it!)

Classes/Types of attacks

- Very coarse classification:
  - Information leakage: Not an actual attack, but allows profiling for one
    - Becomes a real attack in combination with trusting the client/input validation
  - Attacks against cryptography: Typically not breaking, but circumventing it
    - "Adding TLS" is not going to help one bit if the key is static
  - Incorrect code: Forgetting about security or implementing it erroneously
    - Note: The code is perfectly working (=functionality) & might even be tested for this!
    - Good algorithm + bad implementation vs. correct implementation of bad algorithm
  - Trusting the client: Protecting the clients from themselves
    - Plus input validation (see below!)
  - Input validation problems: The trusted user sends data from his/her client computer…
    - … but it turns out to be not that harmless at all (or not that user ☺)!
    - Can be anything: Data, programs/scripts, commands (shell, DB,…) etc.
    - This is the main and most important type!

Difficulties of protecting against attacks (1)

- You need to know about the attacks (at least classes/types) to be able to protect against them
- Most examples and getting started guides are extremely "bad"
  - And this is never even mentioned or corrected (best example: SQL injection)
- You have to protect everything all the time against anyone
  - The attacker can choose, wait, and try again
- Problems are often not easily "localized" (="this is the erroneous statement")
  - Mostly such statements are correct, but should not be used at this locations and while this activity is going on and while the DB is in a certain state (emergence!)
- A defect might not be exploitable because of the system design, but on extension, modification etc. it suddenly becomes so
- You may not be able to fix it: Defect in library (might even be open source!)

Difficulties of protecting against attacks (2)

- Vulnerabilities are often significantly downplayed by vendor
- Whose job is it (would it be) to fix it?
  - Browser vs. plugin, framework vs. OS vs. application, …
- Technological limitations:
  - Some procedures/functions just **cannot** be used securely
  - Protocols or standards might have flaws, but must still be implemented "as defined"
    - Because of various reasons, e.g. compatibility or legal
- Disclosure process varies:
  - 0-day attacks; sent to developer but ignored/delayed; multiple mailing lists; …
- Marketing: "This system is secure!" → But against what in which circumstances?

This does NOT mean its hopeless!

Summary

- Web applications will definitely be attacked – sooner or later
  - Automated software kits used by "script kiddies"
- Getting it secure is very difficult, but "reasonable" security is not hard!
  - A bit of Floriani principle/not-in-my-backyard, however …
- Security must be built in from the beginning and needs at least some monitoring
- As user you cannot depend on the provider
  - Some things he cannot do, some things he won't
- Prepare for incidents or unpatched vulnerabilities
  - Modules for restricting access/filtering URLs etc. should be in there from the start
  - Backups, alternate versions (e.g. static copy of website) etc. should be prepared

Thank you for your attention!

Michael Sonntag

Institute for Information processing and microprocessor technology (FIM)

Johannes Kepler University Linz, Austria

sonntag@fim.uni-linz.ac.at