An Introduction to Information Security: Principles and Practices for Developers

Document Version 3.5
January 2020

NOTE: The coding reference and examples used in this document might become deprecated at some point in time. They serve as examples of secure coding practices in a variety of languages as of 2017. These samples will be periodically reviewed for relevancy but may not serve as best coding practices today.

The latest version of this document is always available at https://allanalford.com/resources

Allan Alford, CISO

https://allanalford.com

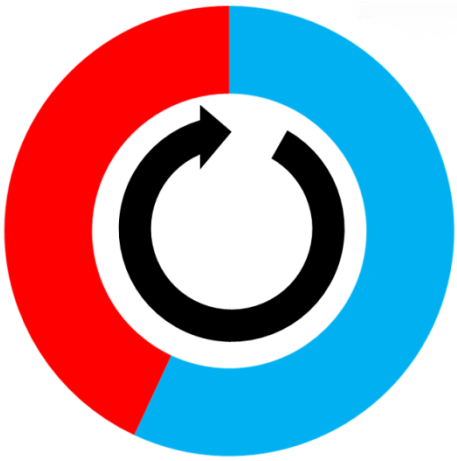
Table of Contents

1. Security Definition and Overview

The most important thing to understand about security is that security is not a set of features, and it is certainly not a *state*. It may exist as an ideal state, but it in the real world, security is best defined as a *process*, one that offers concrete steps, tools, and parameters, and one that is *iterative*.

Security must be iterative, for today's best practices and current software packages are tomorrow's compromised products. There are no guarantees. There is only going through the best possible practices repeatedly, knowing that this way you will at least prevent the majority of security situations before they arise.

The "Security Donut" Model – Perpetual, Iterative Improvement



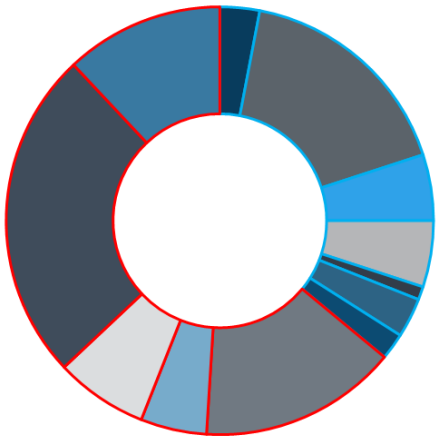
BLUE sections are strategic processes that invest in security. **RED** sections are reactive processes.

Every time we lap around the donut, we invest in more **PROACTIVE** and shrink the percentage of **REACTIVE** effort by a small amount.

Note that the **RED** never goes fully away, as reactive behaviors will always exist.

Reactive processes might include emergency customer calls, bug reports, onboarding CVE's, managing inherited third-party vulnerabilities and defects, failed vulnerability scan reports in the field, etc.

Proactive processes might include creating new security architectures, code reviews, static code analysis, vulnerability scans, patch management policy, firewall in the product, etc.



Monitor and break down your **PROACTIVE** and **REACTIVE** efforts/projects/programs by person hours, by calendar weeks, whatever it takes to see how much time you spend in each category. Analyze the **BLUE** to find your lowest-hanging fruit. Tackle the smallest projects first that give the biggest return on investment.

Each quarter, each year, you should be working to reduce the **RED**.

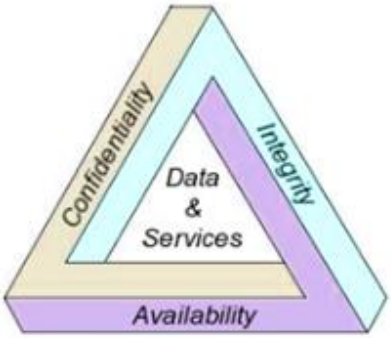
2. Why We Care About Security – C.I.A.

Security is not just about stopping the bad guys. Security is an adjunct to quality. In fact, one cannot have good quality without good security and vice versa. What are some of the maxims for both security and quality?

Neither can be "tested in" after the fact... Both are ongoing processes... Both must acknowledge that perfection is unlikely and/or transitory… And both fundamentally concern themselves with producing the best possible customer experience.

One of the many models used to explain security in terms of customer experiences is "C.I.A." This model declares that data and services are the actual entities that require security, and expresses the security statements in positive, customer-focused terms. This customer-centric view is critical to the whole model.

It is important to note that we are discussing the customers' data and services here, as opposed to whatever data and services we the designers of the system deem to be most vital during the design and delivery of the product. Sometimes key system data or services are indeed of high concern to the customer as well – access to the whole database is access to the customer's specific data in the database, for example. A customer-centric focus lets us consider all things. C.I.A. outlines the three main customer requirements with regards to *their* data and services:



Confidentiality – That which must not be revealed is not revealed to anyone other than the intended people and/or processes, no matter where in the system the need for confidentiality might exist. Examples: phishing attacks violate confidentiality by getting bank information into the wrong hands. "Social engineering" seeks to do the same.

Integrity – Over the course of the full lifecycle, information and configuration should remain unchanged unless changed on purpose by those with proper reason and credentials. Examples: SQL injection attacks violate integrity as do basic website defacement attacks.

Availability – Uptime is not just about sysadmin performance metrics. An unavailable system, service, or set of data can be thought of as compromised. Availability refers to the expected behavior of all subsystems, services and data – not just the system as a whole. Examples: DDoS attacks violate availability as do viruses and other malware.

3. A Few Security Terms

Some Security Definitions

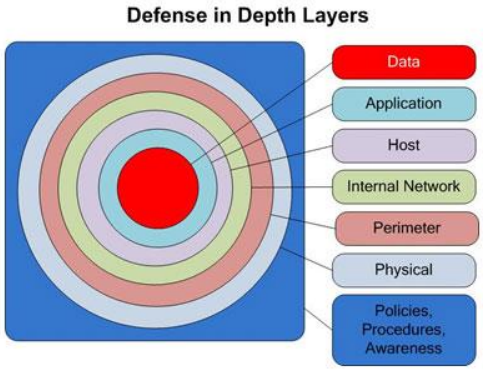
Attack Surface	Your attack surface is absolutely every facet or aspect of your system, service, data etc. that interfaces in any way with any part of the world. A system with a web interface, an API interface, and FTP interface, an SNMP interface all open presents more attack surface than a system with API only. But attack surface can also include human beings, policies, procedures, sticky notes and anything else environmental upon which your system depends in order to perform its intended functions and duties.
Attack Vector	An attack vector is simply a place from whence and attack may happen. *CVSSv2* (Common Vulnerability Scoring System version 2 – see below) is a universal way of assessing vulnerabilities and scoring them to understand severity. CVSS uses attack vectors as part of its severity assessment. In the CVSS system, the three main vectors are: Local (attacker must have physical access or a local account on the vulnerable system), Adjacent Network (attacker must have access to the broadcast or collision domain of the vulnerable system), and Network (the system's vulnerability is operating at Layer 3 or higher on the OSI network model). More real-world attack vectors could be thought of as: access to the data center, local account, LAN, Internet, etc.
CVE	Common Vulnerabilities & Exposures; CVE is the most common method worldwide for identifying and tracking vulnerabilities in the wild. A CVE identifier is issued to a vulnerability once it is confirmed (although sometimes placeholders are granted as well). CVE's can be reported by anyone in the public, but most often are reported by companies with regards to their own products. A self-report is almost always delivered after a fix is available so that the company does not create a liability situation, putting its customers at risk with no fix. CVE's are maintained by MITRE, a contractor to the US Dept. of Homeland Security, and are maintained here: https://cve.mitre.org/
CVSSv2/v3	Common Vulnerability Scoring System, version 2 or 3; CVSSv2 is the most used worldwide method for evaluating a vulnerability and issuing as objective a score as possible. It competes with other methods like DREAD but is in use by nearly every major vendor of network-connected products in the world. CVE's are posted with CVSS scores so that the new vulnerability can be immediately assessed for severity. Note that a CVSS score posted with a CVE for, say, RedHat Linux kernel, might warrant a local CVSS scoring exercise for your own product that features embedded RedHat. Sometimes a generic high score becomes lower (or vice versa) due to compile-time or deployment-time considerations. CVSS is maintained by the Forum for Incident Response & Security Teams (www.first.org) and was jointly developed by many entities both public and private.

Exploit	(Verb) to turn a vulnerability into malicious advantage, compromising a system's security posture, violating C.I.A. (Noun) a method, system, program, or process for turning a vulnerability to such malicious advantage.
Non-Repudiation	The system's ability to prove that a specific user and only that specific user performed an action, sent a message, changed a configuration, etc.
Pen Test	("penetration test") "white hat" hacking, ethical hacking, etc. This is using "black hat" (bad guy) tools and techniques to probe your own system for weaknesses, vulnerabilities, exploits, etc., generally before releasing products to the wild for the first time. The value of in-house pen testing extends to post-GA products, but should be focused primarily on pre-release.
Risk Management	This term exists in most professions, but in security it means weighing the likelihood of a vulnerability being exploited vs. the impact if it is exploited. As with all risk management, a matrix with X and Y axes is used. Details vary, but the point is that sometimes security is about accepting calculated risk and prioritizing other security efforts when it comes to resource commitments.
Threat	What we're protecting against. A threat is anything intentional or accidental that might exploit a vulnerability. Examples: a bitter, ex-IT employee whose credentials are still active, a root console left logged in and unattended, a clear-text password sent via telnet, etc.
Threat Model	Threat modeling is an exercise by which one combines attack surface, attack vectors, understanding of threats, acknowledgement of any vulnerabilities, and then attempts to determine the most likely exploit scenarios and failure points. Threat modeling is a discipline with its own rules, symbols, etc. Threat modeling is a more advanced activity, usually used in conjunction with *pen testing* and *vulnerability scanning*, along with *risk management*.
Threat-Source	NIST SP800-30, a US government security standard, defines this as "Either (1) intent and method targeted at the intentional exploitation of a vulnerability or (2) a situation and method that may accidentally trigger a vulnerability". The accidental variety of threat sources can impact all three categories of C.I.A.
Vulnerability	A flaw in the system that allows for a compromise in the security posture; may be found and never exploited; may be found because it has been exploited.
Vulnerability Scan	An automated scan, usually performed by any number of commercial tools (and a few free tools) in order to start the first pass in a *penetration test*. Useful on its own, it mainly identifies known vulnerable versions of operating systems, popular packages, etc. Vulnerability scanning should always be used but should never be relied upon as the final work as to a system's security posture.

Detailed Security Terms - Defense in Depth

The idea behind defense in depth is very simple: in a complex system, the responsibility to secure resides in more than one locus. In fact, a well-designed defense-in-depth security strategy coordinates security at each concentric ring of attack surface, these rings representing the progressive interfaces between the core of the system to be protected, and the world. In the diagram below, for example, the firewall team has the perimeter covered, but the network team still has IPS. The host has locked down all unnecessary ports, etc. Each team does their part at each layer to ensure best possible security.

The outermost layer in the most effective defense-in-depth models is one of policy, procedure and awareness. This is because other workers – the "good guys" – are usually the most public way into the inner rings. Either by ignorance, neglect, sloppiness, or social engineering, it is the people in an organization who most often let the first wall fall in your security posture. Sound policy governs human behavior (and system behavior as well) to educate, inform, and to hold accountable adherence to sound security policy.



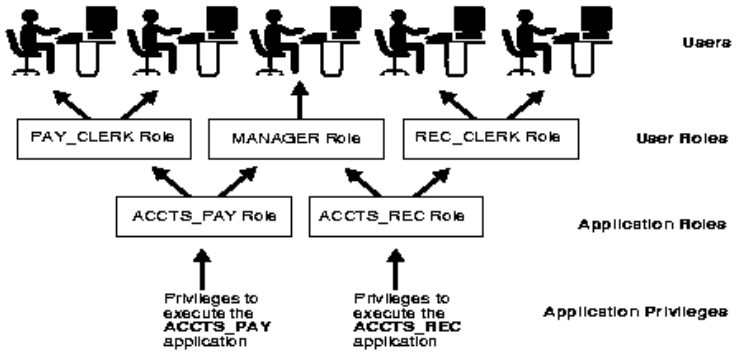
Detailed Security Terms - Least Privilege

Least privilege simply means that an individual or a process is given no more privilege than necessary to perform their job. In the simplest models, this means users are users and admins are admins. More complex systems can have power users, auditors, application administrators, etc. Even more complex systems start with such "buckets" for users and add or remove individual rights, privileges, access, etc.

Some fundamental examples of least privilege:

- Generic users should never have to perform administrative tasks or functions.
- Web servers and other servers should never run as root.
- Privileged accounts should be supplemented in the UI in some way to indicate that a privileged account is logged in – red borders around the screen or other visual cues.

The below diagram demonstrates Least Privilege of both users and applications:



Detailed Security Terms - Secure Defaults

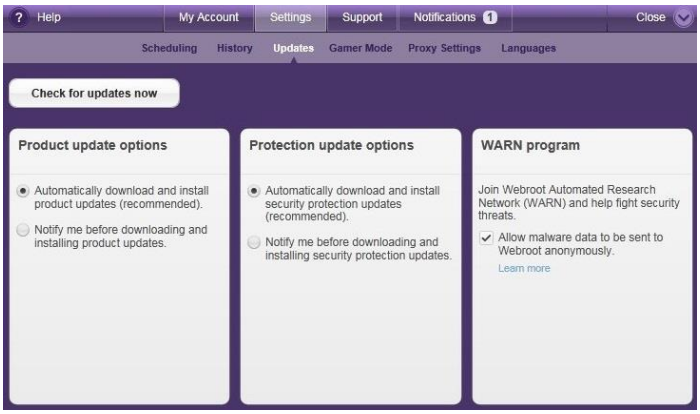
This concept is one of the easiest of all, yet one of the most overlooked in most systems. For every subsystem, process, task, etc. for which there is a more secure option and a least secure option, *always default the system to the most secure option.* The customer may choose to go with a less secure option, but they should have to make a conscious decision to do so, preferably with warnings in the UI explaining the consequences of their choice.

One of the single most important secure defaults is to not ship with any hard-coded credentials for your OS, application, database, etc. ALL hard-coded credentials eventually make it to the Internet, and all can be used to exploit your product. This one cannot be emphasized enough.

More important than that, DO NOT ship with "Easter eggs". Even if they open no ports and do not seem to increase your attack surface from the network perspective, Easter eggs add meaningless, unnecessary attack surface to the local system. NEVER put Easter eggs in professional products.

Another part of secure defaults is sound *patch management* practice. If your product resides on a third-party OS, is your image of that OS patched to current levels at the time you release your product to the market? Does your post-release product offer customers a relatively easy way to keep current over time? The same questions and the same principles of patch management apply to third-party packages, modules and packages from your own company, and even your own development environments. Never release a product with known security issues, vulnerabilities, etc.

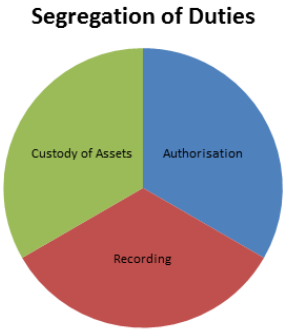
The below diagram demonstrates an out-of-box UI with multiple secure defaults:



Detailed Security Terms – Separation (Also "Segregation") of Duties

At a high level this one refers to the principle of splitting privileges among multiple individuals or systems. This concept obviously hails from environments in which internal personnel are treated with as much or more suspicion (regarding the crafting of security policy) as are external entities. The likelihood of an insider becoming malicious is higher than most realize. The impact of their maliciousness is much higher as they are insiders; passwords, knowledge and access are all in-hand.

Separation of duties seeks to minimize the damage any one insider can do. The classic duality in separation of duties is the administrator and the auditor: The administrator has the power to alter anything in the system, but he cannot delete the logs that showed it was he who made the changes. The auditor can delete logs to cover his own tracks but has no power to make any changes. Both agents must become malicious and collude for any untraceable damage to be done to the system.



4. Understanding Your Customer's Security Concerns

Every customer is different, and every customer has unique security concerns whether they know it or not. The diversity of concerns is so vast that making any assumptions during a customer encounter may result in wasted effort or a lost customer.

Models like CIA help to ask the right questions, but it is also imperative to understand the customer's threat model, risk management matrix, and hierarchy of concern with regards to actors. In simplified terms, are they more worried about nation-state attackers, organized crime, or employees? Does your application process data they consider to be highly sensitive, or is the data lower on their risk register?

Are they willing to carry certain amounts of risk with regards to each possible source of threat, and are those amounts quantified? Many customers do not use these formal security processes and yet are still clear as to what concerns them the most. Ask questions and assume nothing.

5. Benefits of Strategic Security

Securing products up-front is significantly less costly than securing them on the back end – *more so than with other bugs or quality issues*. Lack of security brings with it a more intense damage to the brand, more fervency in dissatisfied customers, and possible liability. Customers will tolerate a lapse in quality far more than a lapse in security.

When security governance offers predictability, accountability and visibility for the security posture of the products, product delivery and support processes operate at reduced costs. An added value is in improving trust with your customers and enhancing your brand reputation.

Uniform security deployment and fix methods speed up delivery to customers while costing fewer cycles within the product development and delivery organizations.

6. Network Security

If your product is network-connected – especially if your product is on the Internet – then the network is most likely your number one source of an attack. As noted above, attacks come in many colors; attempts to steal data, attempts to alter data, attempts to vandalize, attempts to covertly monitor, and attempts to simply shut the system down. Each of these attacks can only take place with one or more attack vectors, and if the attacker is an outsider, that vector is most likely the network.

Minimizing your attack surface, maximizing the security of your default deployment, and knowing your threat model are the key concepts here.

Network Security – IP Configurations

First and foremost, shut down ALL ports that are not needed, including ones that represent optional user behavior. Let the users make the conscious choice to bring ports back up when desired.

A spoofing attack is a type of attack whereby a malicious party impersonates another device or user on a network in order to launch attacks, steal data, or get around access controls. There are a wide variety of spoofing attacks. Some of the most common methods include IP address spoofing attacks, DNS server spoofing attacks, and ARP spoofing attacks. For purposes of securing a product (a single host), IP spoofing is the most critical. The others are best handled elsewhere on the network.

IP spoofing is very simple: A malicious attacker sends IP packets from a false (or “spoofed”) source address in order to disguise itself. The connection between spoofing and DDoS attacks resides in this trick. One type of DDoS attacks uses IP spoofing to overload networks and hosts with packets that appear to be from legitimate source IP addresses. Another technique is when the attacker spoofs the target’s IP address and sends packets from that address to many different legitimate hosts on the

network. When a second-party innocent system receives a packet, it will automatically respond to the "sender", flooding the target's IP address.

IP spoofing attacks can also be used to bypass IP address-based authentication. Therefore, whitelisting firewalls (see below) are only one part of our defense in depth. A two-way certificate exchange from machine-to-machine will foil spoofers in this kind of attack.

Foiling traditional DDoS attacks is a trickier proposition, for a well-crafted DDoS attack simply asks your system to do what it would be doing anyway – responding to legitimate packets. DDoS attacks rarely rely upon anything more than sheer numbers. A robust web server designed to serve 1,000 pages a second will crash when the DDoS botnet pushes it to 10,000 requests a second. To deal with these scenarios, there are some rudimentary techniques.

Using an internal software firewall (iptables or something similar), you can craft a rudimentary anti-DDoS ruleset. Such a tool and the right rules can help foil other basic malicious attacks as well:
Rule #1 – If you can whitelist, do so. A whitelist is a statement that only certain IP's should ever be talking to Port X on your system. All other traffic is automatically denied.

Rule #2 – If you know that the natural behavior of your system is such that its intended peers only ask for a request on a given port every X units of time, observe and block any IP that is hitting the same port at too fast a rate.

Rule #3 – If you normally only have one or a few systems hitting a given port, and you now have hundreds or thousands, see if you can block based on subnet, on route in, or some other means. If you contact the owners of the other systems by human means, you can possibly implement the whitelist in Rule #1.

Rule #4 – If one address, instead of doing business with the expected port or ports, seems to be checking every port on your system, blacklist that IP from all ports. This is an initial probe, and the real attack will come later.

Rule #5 – Blacklist swaths of the Internet you know aren't supposed to interact with your system. No customers or peers in Ukraine? Block the whole country. Most DDoS attacks come from all over, but there is often a heavy bias towards Eastern European, Western Asia and the Middle East in the actual IP src list.

Rule #6 – Most DDoS attacks come from botnets. Most botnets are built when known defects in specific types of systems are exploited en masse. So, the attacking hosts might all be Windows 2000 systems or a specific brand of consumer firewall/router. Develop a means of fingerprinting the attacking systems and block all systems of that type.

Rule #7 – A whitelist corollary to Rule #6 – if only Prysm systems should be chatting with Prysm systems, use fingerprinting in the positive sense – only allow known Prysm systems, and ALL botnets will fail.

There is a concept called "honeypotting" whereby you allow the bad guys to hit the various ports they are trying to hit, but you ensure that all return packets from your system are sent just shy of TCP/IP or UDP timer limits. The incredibly slow responses keep the bad buy latched on, but unproductive. He is stuck with his paw in the honey pot just like a certain famous cartoon bear. This measure is usually

performed for purposes of study – while the bad guy is trapped, you can study his methods, means, and hopefully come to understand his goals.

Here are a few examples from iptables for various purposes:

#1 – Prevent DDoS to the web server:

```
iptables -A INPUT -p tcp --dport 80 -m limit --limit 50/minute --limit-burst 150 -j ACCEPT
```

- "-p tcp --dport 80" means that we are making changes only to the web server listening on port 80
- "–limit 50/minute": This limits connections to a maximum of 50 per minute. Change this value based on your server's specific ability to handle load.
- "–limit-burst 150": The limits-per-minute will be enforced only after the total number of connections has reached this declared limit-burst level.

#2 – Block a specific address altogether:

```
BLOCKED_IP="#.#.#.#"
iptables -A INPUT -s "$BLOCKED_IP" -j DROP
```

#3 – Block only TCP connections, and only to device eth0:

```
iptables -A INPUT -i eth0 -s "$BLOCKED_IP" -j DROP
iptables -A INPUT -i eth0 -p tcp -s "$BLOCKED_IP" -j DROP
```

NOTE: In Windows, *netsh advfirewall* is the alternative to iptables

#1 – Block ICMP Traffic

```
netsh advfirewall firewall add rule name="Block Type 13 ICMP V4"protocol=icmpv4:13,any dir=in action=block
```

#2 – Block an IP Address

```
netsh advfirewall firewall add rule name="disallow suchandsuch webapplication" action=block enable=yes localip=any remoteip=#.#.#.#
```

Also, see the section on Certificates and Cryptography. Mandated crypto connections backed by two-way certificate exchange will foil all manner of malicious activity.

More IP-Level Tricks to Foil Spoofing and DDoS Attacks

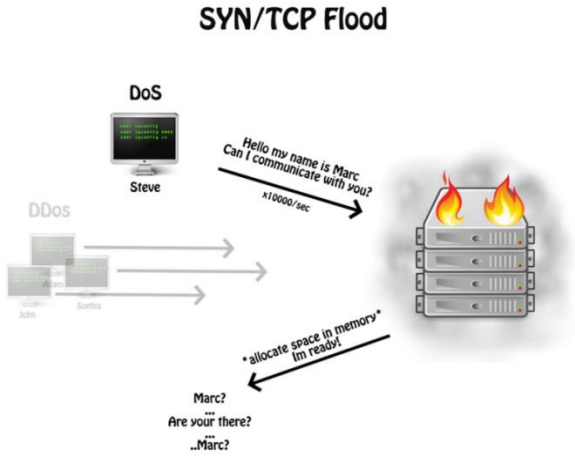
To block ping-based DDoS attacks, simply block ICMP to your host – either at the external firewall (preferred) or inside your internal firewall.

One popular DDoS attack is for the attacker to first find on the Internet any open responder DNS servers, then to send fake DNS queries to those servers, all spoofed with your IP address, and then,

using the conveniently easy to spoof UDP protocol, to sit back and watch all those DNS servers answer your server.

To prevent this attack, block inbound UDP to port 53 from all but planned DNS servers.

A Syn attack ("Syn flood") is a type of attack where the malicious botnet setups up initial TCP connections without then following up – leaving your host waiting on each of a very large number of connections. This is a very TCP-specific version of a DDoS attack. To block Syn attacks, increase your TCP backlog, reduce the Syn-Received timer, or use Syn caches.



Implementing IPSec will help address both spoofing and DDoS attacks.

HTTPS with TLS helps prevent website spoofing.

7. Cryptography & Certificate Best Practices

Since so much of cryptography relies on certificate exchange for identification, and since keys are stored via cryptographic means, this section includes guidance on both cryptography and certificates.

Sound cryptography practice really boils down to a combination of the philosophies behind secure defaults and patch management – know what protocols, algorithms and cipher strengths are insecure and use the stronger options. Create a network of trust with certificates and stick to it.

For every single clear protocol in and out of the system that exists, an encrypted alternative should be available, and should be the default protocol made available out-of-box.

- SHA-256 should be used for one-way hashes, and SHA-1 should be considered deprecated and not even installed in your product. MD5 is deprecated and should not be installed.
- Kerberos should be used for authentication, and NTLM (including NTLMv2) should be considered deprecated.
- SSLv3 is deprecated. TLSv1.2 should be the default with fallback to TLSv1.1 if necessary
- TLSv1.2 should be run at 256-bit strength by default, falling back to 128-bit only if necessary.

- ALL TLS cipher strengths weaker than 128-bit should not even be installed in your product.
- Diffie-Hellman keys should be 2048-bit or higher, as should RSA key sizes.
- Where used, cryptographic hash functions (message digests) should use 224-bit hash size or greater.
- The Prysm metadata should probably be transmitted via AES to complement the media encryption in use with the video codecs with which Prysm operates.

All systems should support PKI for machine-to-machine authentication and for user logons. The machine-to-machine method is a two-way exchange of certificates set up before an encrypted connection is established. Each system offers its identity as validated by a certificate server with a CA that both systems share in a hierarchy, or at least are programmed by default to acknowledge and trust.

It is only after such a certificate exchange occurs that any encrypted connection is open between the two systems. If all connections in which sensitive data might pass are required to be encrypted, and all encryption requires two-way certificate exchange before instantiation, then a significantly greater degree of privacy is assured.

Regarding Certificates:

- The customer should be able to install, view, and remove Root CA and sub-CA certificates.
- The customer should be able to install, view, update and remove server identity certificates and client identity certificates (if they are supported).
- The product must support either OCSP (preferred) or CRL for verifying certificates.
- The product must allow the customer to manage a list of known, trusted certificate authorities.
- Encryption keys must never be stored in clear text
- Audit logs must capture all activities related to key management such as installing, updating, or removing a certificate. All Changes to CRL or OCSP should also be logged.
- The root signing key (used for the signing of software images) must be stored on a computer system that is utterly off the network. Use an “air gap” for this server.
- The signing key itself must be signed by a trusted, public, root authority such as Verisign, Thawte or Entrust. A public CA for your company, itself signed by such an entity, can also be used. The benefit of a 3rd-party, public, trusted CA is that clients (web browsers, etc.) already have them installed. Customers do not have to choose to trust them.

CRL’s (If Used)

- The customer should be able to view a list of all installed CRL’s on the system
- The customer must be able to remove the CRL for any CA certificate that is installed.
- If the customer uses multiple CA’s, the space and time involved in loading CRL’s can become prohibitive very quickly. This is why OCSP is recommended.

8. User Access Security

As mentioned above, PKI support for user logons is desired (inclusion of client identity certificate into the authentication process). A common mechanism in the US DoD, for example, is the CAC, or Common Access Card. The CAC stores a user certificate, the console has a CAC reader, and the logon process now follows the maxim, “something you have, and something you know”.

The “have” part is the certificate – it validates the user’s identity per a trusted CA hierarchy. The “know” part is the traditional username and password. A bad guy can steal your badge, but not know your credentials. Another bad guy can capture your credentials, but not have your badge. These two factors ensure the best possible confidentiality for logons.

CAC readers are ubiquitous, and companies sell keyboards with CAC readers built right in. Microsoft OS’s and browsers support the PKI logons using CAC certificates. Resources on CAC are here:

https://militarycac.com - guides and how-to’s for installing military-approved CAC

https://msdn.microsoft.com/en-us/library/bb742531.aspx – older Microsoft guide on installation

https://technet.microsoft.com/en-us/library/cc775842%28v=ws.10%29.aspx – MS article on CAC and user logon

Password Guidance from OWASP

OWASP recommends the following for password guidance:

- Do not limit maximum length (within reason) or character set of passwords
- Use an encryption with a strong salt (FIPS 140-2 works here)
- Design your password storage with the assumption that it will one day be compromised.
    - Two-factor authentication is a must
    - Once an account is viewed compromised, others can still work if a new protection scheme is loaded and keys are rotated
    - If a user logs on with old credentials, demand 2-factor or deny. Prompt for credential change
- OWASP has more guidance beyond the scope of this Simple IA Model, available here:
- https://www.owasp.org/index.php/Password_Storage_Cheat_Sheet

9. Physical Security

There are three aspects to physical security:

- Physically securing your computers and other main components in your system: There are many vendors who provide physical cabinets for locking up your gear at a customer site. Many of these are based on standard data center rack form factors, and many are based on proprietary need for AV equipment in board rooms. There is similar equipment for securing televisions to walls. Rather than recommend a single vendor, the recommendation is to find ones who are TEMPEST-certified. TEMPEST is not only one seriously sturdy system, but it is certified to block any electrical radiations that could possibly be used to violate C.I.A. This is “the Cadillac” of secure boxes, and others without radiation shielding will certainly suffice in many situations.
- Using physical proximity as part of your security posture: if your main metadata computer and your Cascade are both locked up securely in cabinets (even separate ones), running one single

physical cable between them, not plugged into any switch or network, and having that cable be a visible connection between the two cabinets, can suffice as a physical security argument when concerns about meta data are brought up (in an admittedly unrealistic model). Extending this model to another Cascade means that a dedicated hub or switch in its own cabinet must be used, or VLAN/Layer 3 tagging techniques. Start physical whenever possible and work to a dedicated virtually physical management network in your progression of security concessions in this space.

- Securing the space in which the gear resides. A locked door or a badge reader are a great start but pay attention to artificial ceilings (drop ceilings), and the crawlspaces afforded by raised data center flooring. Securing walls and doors is not enough. Remember, the three ways in are: over, under, or through.

10. Robustness from Application Level Attacks - General & .NET

In general: Update your dev environment whenever updates are available.

In general: Pay attention to deprecated functions/procedures/methods and do not use them.

When developing with environments like .NET, never assume that the client will properly handle input validation. A "middleware" validation module should be written that parses and either clears or rejects all inbound data from *all* user interfaces before allowing it to be passed to key systems (core application, database, etc.) Examples of such inputs could be hidden fields, cookies, URL parameters, form field parameters, DNS results, JavaScript variables, local file systems, etc. Sanitizing all inputs is the most key, as most successful attacks from the outside take advantage of a lapse in input validation.

Get in the habit of using User-ID's with the least privileges possible for the task at hand. Use PrincipalPermission on ASP .NET pages, for example.

Never store sensitive information in cookies and especially not in URL's.

.NET with C# does not have the problems with buffer overflows as do many older development environments, but still gives you the tools to shoot yourself in the foot with regards to memory management.

- DO NOT USE the ***unsafe*** and ***unchecked*** keywords, as these can cause overflows.
- For application-level decisions in dealing with unplanned excess data, see the advice above regarding input validation.
- Also, AVOID UNSAFE API's such as the ***Marshal*** class.
- Memory can be corrupted other ways, so AVOID ***StructLayoutKind.Explicit***.

Code Examples & Config Examples to Foil Top Vulnerabilities

If we compare the OWASP Top 25 vulnerabilities and the CWE/SANS Top 25 Vulnerabilities over a few years' time, we will see a list in which input validation is most important, but in which good coding practices also have a hand in prevention. We revisit some of the topics here due to their importance. All guidance is new in this section:

Classic Buffer Overflow:

- *BAD: fprintf(tracker, trackingdata)* prints the tracker file to a variable trackingdata of unspecified type. Extra data inserted could be used to overflow the buffer
- *GOOD: fprintf(tracker, "%s", trackingdata)* ensures that it prints to a string output only, making the system deal with it appropriately

- Function that restrict the number of bytes prevent unbounded copies, writing outside array bounds, overruns, etc.
    - BAD: *sprintf()* GOOD: *snprintf()*
    - BAD: *strcpy()* GOOD: *strncpy()*
    - BAD: *gets()* GOOD: *fgets()*

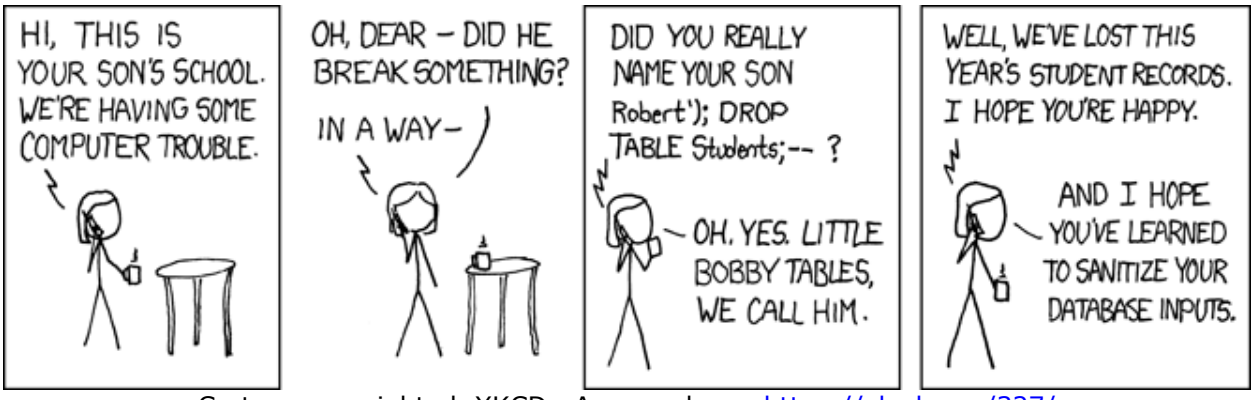
Integer Overflow:

- Integers can overflow as well. Observe the following BAD code. bytesPolled will probably overflow, always creating a lower number than MAXGET. And as a reminder of the malicious side of a buffer overflow, this loop will also overwrite the first MAXGET-1 bytes of the buffer…

```
short int bytesPolled = 0;
char buf[ALARGENUMBER];

while(bytesPolled < MAXGET) {
        bytesPolled += getFromInput(buf+bytesPolled);
}
```

SQL Injection:



Cartoon copyrighted, XKCD. Appears here: https://xkcd.com/327/

- SQL Injection can be even easier than that:
    A website has this coming from a form:
        "SELECT * FROM Students WHERE name = '" + userName + "';"
    The bad guy injects this: ' or 'evil'='evil
    Result is this:
        *SELECT * FROM Students WHERE name = '' OR 'evil'='evil';*

Now our bad guy has listed the entire table out and presumably has usernames and passwords.

Cross-Site Scripting (ASP .NET Example (VB)):

```
' SearchResultsNotSanitized.aspx.vb
Imports System
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls

Public Class BrokenSearchPage Inherits System.Web.UI.Page

Protected userInput As TextBox
Protected commandSearch As Button
Protected labelResult As Label Protected

Sub commandSearch _Click(Source As Object, _ e As EventArgs)

// Do the actual search...
        // …

labelResult.Text="You Searched for: " & userInput.Text

// Display the search results…
// …

End Sub
End Class
```

As we see here, user input is not validated and is echoed back. It is here that a malicious user can insert scripts, HTML, HTTP commands to initiate an attack.

OS Command Injection:

Simple example:

BAD URL has no checks on the back end for input validation:

http://ourimportantproduct/cgi-bin/getData.pl?doc=user1.txt

So the bad guy changes it:

http://ourimportantproduct/cgi-bin/getData.pl?doc=/bin/ls|

And now bad guy has run the 'ls' command.

Simple PHP Example: %3B is the URL encoded version of the semicolon:

http://ourimportantproduct/arandom.php?dir=%3Bcat%20/etc/passwd

Now our bad guy has read our /etc/passwd file…

Examples get more complex, but they all involve the following: Expect more than you are expecting from web forms, from URL's, from cookies, etc. CLEAN those inputs until they are confirmed to be what you expect, or are rejected. HTTP POST is a common place for such attacks.

11. Update and Maintenance Robustness

Software Updates

Software updates should be made available automatically through client pull technology. Systems in the field have a checkbox whereby they poll Prysm for updates and automatically pull down (with automatically installing being a second checkbox). This method is recommended only because we have already discussed digitally signing software images above. Push technology has too many complications with firewalls and other edge devices.

For ultrasecure customers, their systems will not be connected to the Internet. In these cases, the following best practices are recommended: 1) Make new images available on your website as rapidly as possible after release. 2) Provide a means for customers to verify the digital signature – a trusted CA in their list of trusted CA's (Verisign, Thawte, etc.) will do nicely, as will a public CA from your company that is in turn vouchsafed. 3) Create an opt-in email list so that customers receive notification when images are posted. Supplement digital signatures with a SHA has whenever posting images to a website.

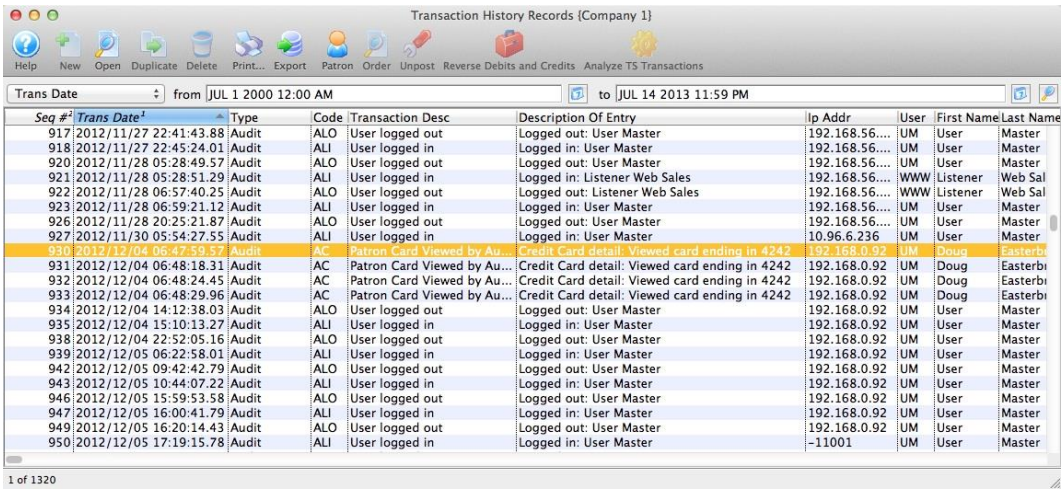
Audit Log Management

Your system needs to provide not just detailed audit logs, but robust audit log management. System logs and debug logs are not the same as an audit log. There are best practices for managing audit logs:

1. The audit log should be separate from the system and debug logs.
2. ALL entries in the log should include date and time as synchronized from a central time server, username or system process that performed the action, details of action performed, any and all warnings that might be needed as a result of the action performed "monitoring capabilities during a call may have been altered". IP address and hostname of client should be logged as well.
3. The audit log should contain ALL logon/logoff information, both successful and failed.
4. The audit log should contain ALL information about changes to sensitive system processes or data.
5. The default out-of-box behavior for logs is that they do NOT auto-revolve, but that the customer can set them to do so. Auto-revolving should be based on lines, bytes, or both.
6. ONLY THE AUDITOR ROLE CAN DELETE, COPY OFF, OR RESET AUDIT LOGS!!!
7. Thus, a trail can be established: Attacker tried to log in three times with Mike's account, and then got in with Bill's old account. He turned off alerts, he turned on call forwarding, and set it

so that all calls will simultaneously forward to a number in the Czech Republic. The IP he did all this from was in the building, however… He was in and out in 5 minutes, logging out manually after trying and failing to delete the audit logs.

The below log contains all the proper elements (assuming failed logons are also captured):



12. Industry Standard Test Tools & Methods

In general, security testing is performed in-house by the good guys with an eye on replicating precisely the sorts of things being done by the bad guys on the outside.

Proper penetration testing takes places in the following stages:

1. Initial scan to identify target ("fingerprinting"). The bad guys use this for fingerprinting, but in-house this first step is usually to validate that what ports are open is the ports that should be open.
2. Initial scan to probe for open ports and to understand the attack surface
3. Initial scan to probe for known weaknesses
4. Penetration attempts based on knowledge gained from the above
5. Attempts to cover one's tracks

Steps 1-3 are generally automated. Steps 4-5 are generally manual. The tools used in 1-3 vary; some are general purpose, and some are tailored to specific operating systems or applications. Note that modern tools often fulfill more than one step in the list above.

In general, here are some great tools, sorted by phase:

1. Nmap (port mapper and more), Nessus (does more)
2. Nmap, IBM Appscan (expensive), Metasploit (same)
3. Nikto (web), Arachni (web), sslscan, testssl, Nessus (general purpose), Qualys Scanguard (expensive), Nexpose (expensive), nCircle IP360 (expensive)

4. Generally a Manual Process
5. Generally a Manual Process

A note on fuzzers: Fuzzers are tools that generate out-of-spec data for inputs (form fields usually): too many characters, the wrong characters, special characters normally reserved for database or other commands… Fuzzers are a great tool as a second pass at a product. But sound, proper and complete input validation (see above) will stop a fuzzer cold every time. If you want to play with fuzzers for web user interfaces, WebScarab, JBroFuzz, and WSFuzzer are all good tools either made or sponsored by OWASP (The Open Web Application Security Project). OWASP are one of the best resources (www.owasp.org) for web-based security concerns.

Static Source Code Analysis Tools

Klocwork is perhaps the most famous tool for analysis of C-based code (and now Java-based as well). It is available with single server and site-wide licensing but is relatively expensive either way. There are plenty of more affordable options, with Checkmarx being recommended rather highly. Latest investigation of the free tools is that they are not up to speed compared to the affordable tier of commercial tools. Yasca appears to be the best of the free tools, operating as a framework that pulls in other tools. However, many false positives resulted from initial testing.

13. Out-of-Box Expected Behaviors

When your product arrives in your customer's hands, this is the moment when the phrase "secure defaults" most applies. How your product behaves when it is first taken out of the box, and how it influences the customer's behavior, will dictate much about the customer's future security posture. Here is a small list of what should happen:

- Your customer is prompted to replace your self-signed (already installed by you) certificate(s) with "real" certificates from their customer environment. They have the option to say, "no", but they must own this decision after being specifically prompted.
- Default passwords for ALL default accounts are prompted to be changed. Again, customer can say, "No", but must own this decision. ALL system passwords are stored in SHA-256 encryption – even database user accounts. NO system passwords are stored programmatically and thus unchangeable.
- As protocols/services/servers are brought up during out-of-box, every one of them should default and prompt to their secure version first (https vs. http, ssh vs. telnet, etc.). Customer again has the option to run insecure but must be the one to actively own that decision.
- In general, no service or protocol should be started, no subsystem brought online, without the customer owning that decisions after a prompt. All prompts should explain the pros and cons of each one.
- Lastly, the customer should be taken to the audit log management interface to see the results of all the out-of-box work – all changes, logins, startups, bringups, shutdowns, choices for insecure

options over secure options, etc. This last piece reinforces the impact of choices made and also ushers the customer into a world of noticing and caring about these things.

14. Recommended Sites

OWASP
http://www.owasp.org

Amazon's AWS Security Center:
http://aws.amazon.com/security/

Cloud Security Alliance:
https://cloudsecurityalliance.org/

FedRAMP
https://www.fedramp.gov/

OWASP's Cloud Top 10
https://www.owasp.org/index.php/Category:OWASP_Cloud_%E2%80%90_10_Project

Forum for Incident Response & Security Teams
http://www.first.org

CVE Page @ Mitre
https://cve.mitre.org

Military CAC
https://militarycac.com/

MSDN guide on installation
https://msdn.microsoft.com/en-us/library/bb742531.aspx

MS Technet on CAC and user logon
https://technet.microsoft.com/en-us/library/cc775842%28v=ws.10%29.aspx

Microsoft Support Article on using *netsh advfirewall*:
https://support.microsoft.com/en-us/kb/947709

Nikto Download:
https://github.com/sullo/nikto

Tenable, the makers of Nessus:
http://www.tenable.com

beyondtrust - Owners of E-eye Retina Scanner: Note: End of Life Just Announced
https://www.beyondtrust.com/vulnerability-management

arachni – Web Scanner:
http://www.arachni-scanner.com/download/

nmap Scanner:
https://nmap.org/

sslcan:
http://sourceforge.net/projects/sslscan/

testssl:
https://testssl.sh/